Fully Homomorphic Encryption over the Integers

Marten van Dijk — MIT
Craig Gentry — IBM Research
Shai Halevi — IBM Research
Vinod Vaikuntanathan — IBM Research

June 8, 2010

Abstract

We describe a very simple "somewhat homomorphic" encryption scheme using only elementary modular arithmetic, and use Gentry's techniques to convert it into a fully homomorphic scheme. Compared to Gentry's construction, our somewhat homomorphic scheme merely uses addition and multiplication over the integers rather than working with ideal lattices over a polynomial ring. The main appeal of our approach is the conceptual simplicity.

We reduce the security of our somewhat homomorphic scheme to finding an approximate integer gcd – i.e., given a list of integers that are near-multiples of a hidden integer, output that hidden integer. We investigate the hardness of this task, building on earlier work of Howgrave-Graham.

1 Introduction

What is the simplest encryption scheme for which one can hope to achieve security? The Caesar cipher is simple, but not secure. We believe that conventional public-key encryption schemes with modular exponentiations are secure, but modular exponentiation is not a very simple operation. If we were to forget our current schemes and start from scratch, perhaps something like the following scheme would be a good candidate for a simple symmetric encryption scheme:

KeyGen: The key is an odd integer, chosen from some interval $p \in [2^{\eta-1}, 2^{\eta})$.

Encrypt(p, m): To encrypt a bit $m \in \{0, 1\}$, set the ciphertext as an integer whose residue mod p has the same parity as the plaintext. Namely, set $c = pq + 2r + m$, where the integers q, r are chosen at random in some other prescribed intervals, such that $2r$ is smaller than $p/2$ in absolute value.

Decrypt(p, c): Output $(c \bmod p) \bmod 2$.

When the noise r is sufficiently smaller than the secret key p, this simple encryption scheme turns out to be "somewhat homomorphic" in the sense of [6] – namely, it can be used to evaluate low-degree polynomials over encrypted data. Moreover, with a judicious choice of parameters (say $r \approx 2^{\sqrt{\eta}}$ and $q \approx 2^{\eta^3}$), this simple scheme may even be secure!

Specifically, we reduce the security of the scheme to the hardness of the approximate integer greatest common divisors (approximate GCD) problem which, roughly speaking, says that it is hard to recover p given the x_i's. This problem, for the case of two x_i's, was analyzed by Howgrave-Graham [11]. Our parameters – in particular, the large size of the q_i's – are designed to avoid a generalized version of his attack (as well as various other attack avenues, such as solving the associated simultaneous Diophantine approximation problem).

So far we only described a symmetric scheme, but turning it into a public key encryption scheme is easy: The public key consists of many "encryptions of zero", namely integers $x_i = q_i \cdot p + 2r_i$ where q_i, r_i are chosen from the same prescribed intervals as above. Then to encrypt a bit m, the ciphertext is essentially set as m plus a subset sum of the x_i's.

As described, the scheme supports a limited number of additions and multiplications over encrypted bits. However, it is easy to see that the scheme is not *fully* homomorphic: as-is, it cannot evaluate high-degree polynomials over the encrypted data. However, we show in this work that this simple scheme is amenable to Gentry's "blueprint" for constructing a fully homomorphic scheme out of certain somewhat homomorphic schemes [7]. Namely we can "squash the decryption circuit" to get a bootstrappable scheme, and then invoke Gentry's bootstrapping theorem to obtain a fully homomorphic public-key encryption scheme.

We stress that our main motivation is conceptual simplicity – namely, to demonstrate that even something as complex as fully homomorphic encryption can be achieved using "elementary" techniques.

1.1 Prior Work

Variants of the simple scheme that we described above have appeared in prior work, but the full potential of their homomorphic properties was never realized. Here, we discuss two of these results, and relate them to the current work.

First, in a discussion thread on the `cypherpunks` mailing list from 2000, Bram Cohen proposed a public key encryption scheme [4] that is quite similar to the scheme above. Secondly, in 2008, Levieil and Naccache applied the same general technique to construct what they called an *"Insecure and Clumsy" Cryptographic Test Correction* scheme, and used the *additive* homomorphic properties of the scheme to prevent students from cheating on their exams [17, Sec. 3]. Both these works appeared prior to the breakthrough result of Gentry in 2009 that showed the *first* fully homomorphic encryption scheme, and as such, do not observe or utilize the multiplicative homomorphism that the scheme could support. We point out the two main distinctions of our work below.

1. First, the security analysis in these prior works was informal, and concrete parameters were either not set at all, or set to trivially breakable values. The scheme in [17] is trivially broken when considered as a cryptographic scheme, irrespective of the choice of parameters. This is justified in their case since the adversary model they considered is very weak.

   In fact, prior to our work there was widespread belief in the cryptographic community that schemes of this form are inherently insecure, due to the attacks that we describe in Section 5. Hence, one of the contributions of our work is to point out that with an appropriate choice of parameters, this simple scheme *can be made to resist all known attacks.*

2. Second, and more importantly, neither of the works mentioned above even considered multiplicative homomorphism, and specific instantiations (when given) did not support even a single multiplication. Thus, another contribution of this work is to observe that not only can this scheme made to support multiplications, but it can be used within Gentry's blueprint to construct a fully homomorphic encryption scheme.

3. Finally, we exhibit a search-to-decision reduction which shows that the semantic security of the scheme above can be based on a well-defined *search* problem, namely the approximate GCD problem (see Section 4 for more details).

A third cryptosystem which is superficially similar to ours is Regev's first encryption scheme [22] based on the unique shortest vector problem. In fact, a slight variation of Regev's scheme can be described by exactly the same formula as ours, $\mathsf{Enc}(p, m) = qp + 2r + m$. One major difference between these schemes is that the secret key p in our scheme is an integer, whereas in Regev's scheme, the secret key is an integral fraction of the domain size (i.e., $p = N/h$ for a public N and a secret h). A consequence of this difference is that Regev's scheme does not offer multiplicative homomorphism. Another important difference is that our choice of parameters is much more aggressive than Regev's. Unfortunately, this means that Regev's worst-case to average-case security reductions from [22] do not seem to apply to our scheme.

2 Preliminaries

Below we usually denote parameters by Greek letters (e.g., η, γ, τ, etc.), with λ always denoting the security parameter. Real numbers and integers are denoted by lowercase English letters (p, q, x, y, etc.). All logarithms in the text are base-2 unless stated otherwise.

For a real number z, we denote by $\lceil z \rceil$, $\lfloor z \rfloor$, $\lfloor z \rceil$ the rounding of a up, down, or to the nearest integer. Namely, these are the unique integers in the half open intervals $[z, z+1)$, $(z-1, z]$, and $(z - \frac{1}{2}, z + \frac{1}{2}]$, respectively.

For a real number z and an integer p, we use $q_p(z)$ and $r_p(z)$ to denote the quotient and remainder of z with respect to p, namely $q_p(z) \stackrel{\text{def}}{=} \lfloor z/p \rceil$ and $r_p(z) \stackrel{\text{def}}{=} z - q_p(z) \cdot p$. (Note that $r_p(z) \in (-p/2, p/2]$.) We also denote the remainder by $[z]_p$ or $(z \bmod p)$, we use these three notations interchangeably throughout the paper.

A family $\mathcal{H}$ of hash functions from X to Y, both finite sets, is said to be 2-universal if for all distinct $x, x' \in X$, $\Pr_{h \xleftarrow{\text{R}} \mathcal{H}}[h(x) = h(x')] = 1/|Y|$. A distribution D is ϵ-uniform if its statistical distance from the uniform distribution is at most ϵ, where the statistical difference between two distributions D_1, D_2 over a finite domain X is $\frac{1}{2} \sum_{x \in X} |D_1(x) - D_2(x)|$.

Lemma 2.1 (Simplified Leftover Hash Lemma [10])**.** *Let $\mathcal{H}$ be a family of 2-universal hash functions from X to Y. Suppose that $h \xleftarrow{\text{R}} \mathcal{H}$ and $x \xleftarrow{\text{R}} X$ are chosen uniformly and independently. Then, $(h, h(x))$ is $\frac{1}{2}\sqrt{|Y|/|X|}$-uniform over $\mathcal{H} \times Y$.*

2.1 Homomorphic Encryption

Our definitions are adapted from Gentry [7]. Below we only consider encryption schemes that are homomorphic with respect to boolean circuits consisting of gates for addition and multiplication mod 2. (Considering only bit operations also means that the plaintext space of the encryption schemes that we consider is limited to $\{0, 1\}$.) See the works of Ishai and Paskin [12] for a more general definitional treatment of homomorphic encryption with respect to other forms of "programs."

A homomorphic public key encryption scheme $\mathcal{E}$ has four algorithms: the usual KeyGen, $\mathsf{Encrypt}$, and $\mathsf{Decrypt}$, and an additional algorithm $\mathsf{Evaluate}$. The algorithm $\mathsf{Evaluate}$ takes as input a public key pk, a circuit C, a tuple of ciphertexts $\vec{c} = \langle c_1, \ldots, c_t \rangle$ (one for every input bit of C), and outputs another ciphertext c.

Definition 2.2 (Correct Homomorphic Decryption)**.** The scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Evaluate})$ is *correct* for a given t-input circuit C if, for any key-pair $(\mathrm{sk}, \mathrm{pk})$ output by $\mathsf{KeyGen}(\lambda)$, any t plaintext bits $m_1, \ldots, m_t$, and any ciphertexts $\vec{c} = \langle c_1, \ldots, c_t \rangle$ with $c_i \leftarrow \mathsf{Encrypt}_{\mathcal{E}}(\mathrm{pk}, m_i)$, it is the case that:

$$\mathsf{Decrypt}\,(\mathrm{sk},\ \mathsf{Evaluate}(\mathrm{pk}, C, \vec{c})) = C(m_1, \ldots, m_t)$$

Definition 2.3 (Homomorphic Encryption)**.** The scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Evaluate})$ is homomorphic for a class $\mathcal{C}$ of circuits[1] if it is correct for all circuits $C \in \mathcal{C}$. $\mathcal{E}$ is *fully homomorphic* if it is correct for all boolean circuits.

The semantic security of a homomorphic encryption scheme is defined in the usual way [9], without reference to the Evaluate algorithm. (Indeed Evaluate is a public algorithm with no secrets.)

It is clear that as defined above, fully homomorphic encryption can be trivially realized from any secure encryption scheme, by an algorithm Evaluate that simply attaches a description of the circuit C to the ciphertext tuple, and a Decrypt procedure that first decrypts all the ciphertexts and then evaluates C on the corresponding plaintext bits. Two properties of homomorphic encryption that rule out this trivial solution are *circuit-privacy* and *compactness*.

Circuit privacy roughly means that the ciphertext generated by Evaluate does not reveal anything about the circuit that it evaluates beyond the output value of that circuit, even for someone who knows the secret key. We discuss circuit privacy in Appendix C. It is folklore that circuit-private fully-homomorphic encryption can be realized using Yao's "garbled circuits" [23, 18] and a two-flow oblivious transfer protocol. (This construction is similar to the trivial solution from above, essentially it replaces the plaintext circuit with a garbled circuit.) Hence the "real challenge" in constructing fully homomorphic encryption comes from the compactness property, which essentially means that the size of the ciphertext that Evaluate generates does not depend on the size of the circuit C.

Definition 2.4 (Compact Homomorphic Encryption)**.** The scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Evaluate})$ is *compact* if there exists a fixed polynomial bound $b(\lambda)$ so that for any key-pair $(\mathrm{sk}, \mathrm{pk})$ output by $\mathsf{KeyGen}(\lambda)$, any circuit C and any sequence of ciphertext $\vec{c} = \langle c_1, \ldots, c_t \rangle$ that was generated with respect to pk, the size of the ciphertext $\mathsf{Evaluate}(\mathrm{pk}, C, \vec{c})$ is not more than $b(\lambda)$ bits (independently of the size of C).

2.2 Bootstrappable Encryption

Following Gentry [7], we construct homomorphic encryption for circuits of any depth from one that is capable of evaluating just a little more than its own decryption circuit.

Definition 2.5 (Augmented Decryption Circuits)**.** Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Evaluate})$ be an encryption scheme, where decryption is implemented by a circuit that depends only on the security parameter.[2]

[1] Formally, $\mathcal{C}$ is an ensemble, parametrized by the security parameter.

[2] This in particular means that for a fixed value of the security parameter, the size of the secret key is always the same, and similarly all the ciphertexts that can be decrypted have the same size.

For a given value of the security parameter λ, the set of augmented decryption circuits consists of two circuits, both take as input a secret key and two ciphertexts: One circuit decrypts both ciphertexts and adds the resulting plaintext bits mod 2, the other decrypts both ciphertexts and multiplies the resulting plaintext bits mod 2. We denote this set by $D_{\mathcal{E}}(\lambda)$.

Definition 2.6 (Bootstrappable Encryption)**.** Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Evaluate})$ be a homomorphic encryption scheme, and for every value of the security parameter λ let $\mathcal{C}_{\mathcal{E}}(\lambda)$ be a set of circuits with respect to which $\mathcal{E}$ is correct. We say that $\mathcal{E}$ is *bootstrappable* if $D_{\mathcal{E}}(\lambda) \subseteq \mathcal{C}_{\mathcal{E}}(\lambda)$ holds for every λ.

Theorem 2.7 ([7])**.** *There is an (efficient, explicit) transformation that given a description of a bootstrappable scheme $\mathcal{E}$ and a parameter $d = d(\lambda)$, outputs a description of another encryption scheme $\mathcal{E}^{(d)}$ such that:*

1. *$\mathcal{E}^{(d)}$ is compact (in particular the* Decrypt *circuit in $\mathcal{E}^{(d)}$ is identical to that in $\mathcal{E}$), and*

2. *$\mathcal{E}^{(d)}$ is homomorphic for all circuits of depth up to d.*

Moreover, $\mathcal{E}^{(d)}$ is semantically secure if $\mathcal{E}$ is: Any attack with advantage ε against $\mathcal{E}^{(d)}$ can be converted into an attack with similar complexity against $\mathcal{E}$ with advantage at least $\varepsilon/\ell d$, where ℓ is the length of the secret key in $\mathcal{E}$.

We also note that if the bootstrappable scheme $\mathcal{E}$ is "circular secure" then it can be converted into a single compact fully-homomorphic encryption scheme $\mathcal{E}'$. See [7] for details.

3 A Somewhat Homomorphic Encryption Scheme

Parameters. The construction below has many parameters, controlling the number of integers in the public key and the bit-length of the various integers. Specifically, we use the following four parameters (all polynomial in the security parameter λ):

γ is the bit-length of the integers in the public key,

η is the bit-length of the secret key (which is the hidden approximate-gcd of all the public-key integers),

ρ is the bit-length of the noise (i.e., the distance between the public key elements and the nearest multiples of the secret key), and

τ is the number of integers in the public key.

These parameters must be set under the following constraints:

- $\rho = \omega(\log \lambda)$, to protect against brute-force attacks on the noise;
- $\eta \geq \rho \cdot \Theta(\lambda \log^2 \lambda)$, in order to support homomorphism for deep enough circuits to evaluate the "squashed decryption circuit" (cf. Sections 3.2 and 6.2);
- $\gamma = \omega(\eta^2 \log \lambda)$, to thwart various lattice-based attacks on the underlying approximate-gcd problem (cf. Section 5);

- $\tau \geq \gamma + \omega(\log \lambda)$, in order to use the leftover hash lemma in the reduction to approximate gcd (cf. Lemma 4.3).

We also use a secondary noise parameter $\rho' = \rho + \omega(\log \lambda)$. A convenient parameter set to keep in mind is $\rho = \lambda$, $\rho' = 2\lambda$, $\eta = \tilde{O}(\lambda^2)$, $\gamma = \tilde{O}(\lambda^5)$ and $\tau = \gamma + \lambda$. (This setting results in a scheme with complexity $\tilde{O}(\lambda^{10})$.)

For a specific (η-bit) odd positive integer p, we use the following distribution over γ-bit integers:

$$\mathcal{D}_{\gamma,\rho}(p) \quad = \quad \left\{ \text{choose } q \xleftarrow{\$} \mathbb{Z} \cap [0,\ 2^\gamma/p),\ r \xleftarrow{\$} \mathbb{Z} \cap (-2^\rho,\ 2^\rho)\ :\ \text{output } x = pq + r \right\}$$

This distribution is clearly efficiently sampleable.

3.1 The Construction

$\mathsf{KeyGen}(\lambda)$**.** The secret key is an odd η-bit integer: $p \xleftarrow{\$} (2\mathbb{Z}+1) \cap [2^{\eta-1}, 2^\eta)$.

For the public key, sample $x_i \xleftarrow{\$} \mathcal{D}_{\gamma,\rho}(p)$ for $i = 0, \ldots, \tau$. Relabel so that x_0 is the largest. Restart unless x_0 is odd and $r_p(x_0)$ is even. The public key is $\mathrm{pk} = \langle x_0, x_1, \ldots, x_\tau \rangle$.

$\mathsf{Encrypt}(\mathbf{pk}, m \in \{0,1\})$**.** Choose a random subset $S \subseteq \{1, 2, \ldots, \tau\}$ and a random integer r in $(-2^{\rho'},\ 2^{\rho'})$, and output $c \leftarrow \left[m + 2r + 2\sum_{i\in S} x_i\right]_{x_0}$.

$\mathsf{Evaluate}(\mathbf{pk}, C, c_1, \ldots, c_t)$**.** Given the (binary) circuit $C_\mathcal{E}$ with t inputs, and t ciphertexts c_i, apply the (integer) addition and multiplication gates of $C_\mathcal{E}$ to the ciphertexts, performing all the operations over the integers, and return the resulting integer.

$\mathsf{Decrypt}(\mathbf{sk}, c)$**.** Output $m' \leftarrow (c \bmod p) \bmod 2$.

Remark 3.1. Recall that $(c \bmod p) = c - p \cdot \lfloor c/p \rceil$, and as p is odd we can instead decrypt using the formula $m' \leftarrow [c - \lfloor c/p \rceil]_2 \ = \ (c \bmod 2) \oplus (\lfloor c/p \rceil \bmod 2)$.

Remark 3.2. Originally, we described encryption as adding m to a random subset sum of "encryptions of zero". Indeed, the scheme can viewed this way. Let $w_i = [2x_i]_{x_0}$ for $i = 1, \ldots, \tau$. Each w_i, and also x_0, is essentially an encryption of zero; its noise is even. Moreover, $c = m + 2r + \sum_{i\in S} w_i - k \cdot x_0$ for some integer k.

3.2 Correctness

Permitted Circuits and Polynomials. For a mod-2 arithmetic circuit (composed of mod-2 Add and Mult gates), we consider its generalization to the integers, i.e., the same circuits with the Add and Mult gates applied to integers rather than to bits. Similar to Gentry [7], we define a *permitted circuit* as one where for any $\alpha \geq 1$ and any set of integer inputs all less than $2^{\alpha(\rho'+2)}$ in absolute value, it holds that the generalized circuit's output has absolute value at most $2^{\alpha(\eta-4)}$. Let $C_\mathcal{E}$ denote the set of permitted circuits. Clearly, we have:

Lemma 3.3. *The scheme from above is correct for $C_\mathcal{E}$.*

Proof. The straightforward proof is in Appendix A. □

Remark 3.4. Since "fresh" ciphertexts output by Encrypt have noise at most $2^{\rho'+2}$, the ciphertext output by Evaluate applied to a permitted circuit has noise at most $2^{\eta-4} < p/8$. The bound $2^{\eta-2} < p/2$ would suffice for correct decryption. But we will later use the fact that the noise remains below $p/8$ in Section 6 to perform the decryption operation using a very shallow arithmetic circuit.

The definition of the set $\mathcal{C}_\mathcal{E}$ from above is rather indirect. In particular this definition does not give a good picture of what $\mathcal{C}_\mathcal{E}$ "looks like". By the triangle inequality, a k-fan-in Add gate clearly increases the magnitude of the integers by at most a factor of k. However, a 2-fan-in Mult gate may *square* the magnitude of the integers – i.e., double their bit-lengths. So, clearly, the main bottleneck is the *multiplicative depth* of the circuit, or the *degree* of the multivariate polynomial computed by the circuit. We have the following lemma.

Lemma 3.5. *Let C be a boolean circuit with t inputs, and let $C^\dagger$ be the associated integer circuit (where boolean gates are replaced with integer operations). Let $f(x_1, \ldots, x_t)$ be the multivariate polynomial computed by $C^\dagger$; let d be its degree. If $|\vec{f}| \cdot (2^{\rho'+2})^d \le 2^{\eta-4}$ (where $|\vec{f}|$ is the l_1 norm of the coefficient vector of f) then $C \in \mathcal{C}_\mathcal{E}$.* □

In particular, $\mathcal{E}$ can handle f as long as

$$d \le \frac{\eta - 4 - \log|\vec{f}|}{\rho' + 2} \tag{1}$$

Below we refer to polynomials that satisfy Equation (1) as *permitted polynomials* and we denote by $\mathcal{P}_\mathcal{E}$ the set of permitted polynomials and by $C(\mathcal{P}_\mathcal{E})$ the set of circuits that compute them. The discussion above implies that $C(\mathcal{P}_\mathcal{E}) \subseteq \mathcal{C}_\mathcal{E}$.

Remark 3.6. For our purposes, we consider settings where $\log|\vec{f}|$ is small in relation to η, $\rho' = \omega(\log\lambda)$ and $t, \tau \le \lambda^\beta$, and we need to support polynomials of degree up to $\alpha\lambda\log^2\lambda$ (for some constants α, β). Plugging these expressions in Equation (1), it is sufficient to set $\eta = \rho' \cdot \Theta(\lambda\log^2\lambda)$.

3.3 Variations and Optimizations

3.3.1 Modular-reduction during Evaluate.

Note that while Encrypt reduces the ciphertext modulo the public key element x_0, we cannot do the same in Evaluate. The reason is that after just one multiplication the ciphertext becomes much larger than x_0, so modular reduction will include a large multiple of x_0 hence introducing intolerable error.

To reduce the ciphertext size during Evaluate, we can add to the public key more elements of the form $x'_i = q'_i p + 2r'_i$ where the r'_i's are chosen as usual from the interval $(-2^\rho, 2^\rho)$ but the q_i's are chosen much larger than for the other public key elements. Specifically, for $i = 0, \ldots \gamma$, we set:

$$q'_i \xleftarrow{\$} \mathbb{Z} \cap [2^{\gamma+i-1}/p,\ 2^{\gamma+i}/p), \quad r'_i \xleftarrow{\$} \mathbb{Z} \cap (-2^\rho, 2^\rho), \quad x'_i \leftarrow 2(q'_i \cdot p + r'_i),$$

thus getting $x'_i \in [2^{\gamma+i},\ 2^{\gamma+i+1}]$. During Evaluate, every time we have a ciphertext that grows beyond 2^γ, we reduce it first modulo x'_γ, then modulo $x'_{\gamma-1}$, and so on all the way down to x'_0, at which point we again have a ciphertext of bit-length no more than γ.

Recall that a single operation at most doubles the bit-length of the ciphertext. Hence after any one operation the ciphertext cannot be larger than $2x'_\gamma$, and therefore the sequence of modular reductions involves only small multiples of the x_i''s, which means that it only adds a small amount of noise. (We note that in addition to smaller ciphertexts, this optimization also reduces the public key size when we use the "decryption squashing" technique as described in Section 6.1.)

It is not clear to what extent adding these larger integers to the public key influences the security of the scheme. It does change the specifics of the approximate-GCD assumption that we need to make, but the same decision-to-search reduction from Section 4 still goes through.[3] Also, we note that having integers with these very large quotients does not seem to help in any of the attacks on approximate-GCD that we considered.

Remark 3.7. Note that when using the original scheme without the optimization, homomorphic evaluation of different circuits that compute the same polynomial would result in the exact same output ciphertext (i.e., the polynomial applied to the input ciphertexts over the integers). This is no longer true when using the size-reduction optimization, because of the additional modular reduction steps. For example, evaluating the circuit "$x_1(x_2 + x_3)$" is likely to yield a different ciphertext than the circuit "$x_1x_2 + x_1x_3$."

In principle, it is plausible that evaluating one circuit would yield a ciphertext with small enough noise to be decrypted, while evaluating another circuit for the same polynomial will produce a ciphertext with too much noise. Adapting the "bootstrappability analysis" from Section 6.2 to the optimized scheme, one would have to take into account not only the degree of the polynomial implementing the decryption process but also the particular circuit that implements this polynomial. It should not be hard to argue that the circuit in Section 6.2 does not introduce too much noise, but the analysis is quite tedious and is omitted here.

3.3.2 An even slightly simpler scheme.

One variation of our scheme is to choose the integer x_0 in the public key not as a near multiple of p, but as an exact multiple. This choice allows reducing modulo x_0 also during $\mathsf{Evaluate}$, and also simplifies the analysis from Section 6.2 and the decision-to-search reduction from Section 4.

Although the resulting approximate-GCD problem appears to be "clearly easier" (since the attacker is given one multiple of p with no noise), we do not know of ant attack that works for this "easier case" but not for the general case.

3.3.3 Ciphertext compression.

Even though the optimizations from above keep evaluated ciphertexts at the same length as original ciphertexts, the size of these ciphertexts is still very large – $\tilde{\theta}(\lambda^5)$ bits under our suggested parameters. We next show how to "compress", or post-process the ciphertexts, down to (asymptotically) the size of an RSA modulus, reducing the communication complexity of our scheme dramatically.

The price of this optimization, however, is that we cannot evaluate anything on these compressed ciphertexts. Hence we can only use this compression technique on the final output ciphertexts, after all applications of the $\mathsf{Evaluate}$ algorithm have been completed. (This technique also introduces another hardness assumption, similar to the ϕ-hiding assumption of Cachin et al. [3].)

Roughly, we supplement the public key with the description of a group G and an element $g \in G$ whose order is a multiple of the secret key p. Then, given the ciphertext c from our scheme, the compressed ciphertext is simply $c^* \leftarrow g^c$. Note that $\mathrm{DL}_g(c^*) = c \pmod{p}$, so decrypting is done

[3] Allowing this reduction to go through is the reason that the x_i''s are set as even integers.

by first computing $y \leftarrow \mathrm{DL}_g(c^*) \bmod p$, and then $m \leftarrow y \bmod 2$. Correctness follows immediately from the correctness of the original scheme.

To implement this idea, we need to choose the secret key p as a smooth number so that we can compute $(\mathrm{DL}_g(c^*) \bmod p)$ on decryption. It seems sufficient to choose the secret key as a product of random distinct $\lambda^2/\log\lambda$ small primes (say, all smaller than λ^3). Also, we need to ensure that publishing G, g does not violate the security of the scheme. This can be accomplished by publishing an RSA modulus N such that $p|\phi(N)$ (and $\log N$ sufficiently larger than $4\log p$),[4] along with a random element $g \in_R Z_N^*$, relying on a variant of the ϕ-hiding assumption [3]. Namely, we assume that given two smooth numbers p_1, p_2 as above and given N such that one of the p_i's divides $\phi(N)$, it is hard to determine which of the two p_i's divides $\phi(N)$. In Appendix D we describe this optimization in more details, and provide a proof of security for it under this ϕ-hiding variant.

4 Security of the Somewhat Homomorphic Scheme

We reduce the security of the scheme from Section 3 to the hardness of the approximate-gcd problem. Namely, given a set of integers $x_0, x_1, \ldots, x_\tau$, all randomly chosen close to multiples of a large integer p, find this "common near divisor" p.

On a high level, our reduction resembles classical hard-core-bit proofs in factoring-based cryptography (e.g., Alexi et al. [1]): Fixing a randomly-chosen public key, we roughly show that an adversary who can predict the encrypted bit in a random ciphertext under this public key can be used to find the secret key (for this fixed public key). As in [1], we describe a random-self-reduction and accuracy-amplification step that uses the promised adversary to get a reliable oracle for the least-significant bit, and then a binary-GCD algorithm that uses that reliable oracle to find p.

The technical details, of course, are very different than in factoring-based cryptography. Perhaps the main difference is that our random self-reduction entails a loss in parameters. Specifically, we show that a noticeable advantage in guessing the encrypted bit in a random "high noise ciphertext" – where the noise is ρ' bits – can be converted into the ability to predict reliably the parity bit of the quotient in an arbitrary "low noise integer" – where the noise is ρ bits. (Roughly, the reason for this is that we need to add extra noise to "wipe out the traces" of the non-random noise in the arbitrary input integer.)

The implication is that we can only reduce the security of our cryptosystem in the "high-noise regime" to the hardness of approximate-gcd in the "low-noise regime." Note that the difference between "high noise" and "low noise" is rather small: only $\omega(\log\lambda)$ bits.

4.1 Reduction to Approximate-GCD

The approximate-gcd problem is defined as follows:

Definition 4.1 (Approximate GCD)**.** The (ρ, η, γ)-approximate-gcd problem is: given polynomially many samples from $\mathcal{D}_{\gamma,\rho}(p)$ for a randomly chosen η-bit odd integer p, output p.

Theorem 4.2. *Fix the parameters $(\rho, \rho', \eta, \gamma, \tau)$ as in the Somewhat Homomorphic Scheme from Section 3 (all polynomial in the security parameter λ).*

[4] The condition $\log N > 4\log p$ is needed, since otherwise we can use Coppersmith's method [5] to break the corresponding ϕ-hiding assumption.

Any attack $\mathcal{A}$ with advantage ε on the encryption scheme can be converted into an algorithm $\mathcal{B}$ for solving (ρ, η, γ)-approximate-gcd with success probability at least $\varepsilon/2$. The running time of $\mathcal{B}$ is polynomial in the running time of $\mathcal{A}$, and in λ and $1/\epsilon$.

Proof Recall that we use $q_p(z)$ and $r_p(z)$ to denote the quotient and remainder of z with respect to p, hence $z = q_p(z) \cdot p + r_p(z)$. Let $\mathcal{A}$ be an attacker against the scheme. Namely, $\mathcal{A}$ takes as input a public key and a ciphertext (as produced by KeyGen and Encrypt of our scheme), and outputs the correct plaintext bit with probability $\frac{1}{2} + \epsilon$ for some noticeable ϵ. (The probability is over KeyGen and Encrypt, as well as the choice of the plaintext bit and the internal randomness of $\mathcal{A}$.)

We use $\mathcal{A}$ to construct a solver $\mathcal{B}$ for approximate-gcd with parameters ρ, η, γ. For a randomly chosen η-bit odd integer p, the solver $\mathcal{B}$ has access to as many samples from $\mathcal{D}_{\gamma,\rho}(p)$ as it needs, and the goal is to find p.

Step 1: Creating a public key. The solver $\mathcal{B}$ begins by constructing a public key for the scheme. $\mathcal{B}$ draws $\tau + 1$ samples $x_0, \ldots, x_\tau \xleftarrow{\$} \mathcal{D}_{\gamma,\rho}(p)$. It relabels so that x_0 is the largest. It restarts unless x_0 is odd. $\mathcal{B}$ then outputs a public key $\mathrm{pk} = \langle x_0, x_1, \ldots, x_\tau \rangle$. Clearly, if $r_p(x_0)$ *happens* to be even then the distribution induced on the public key is identical to that of the scheme.

Step 2: A subroutine for high-accuracy LSB predictor. Next, $\mathcal{B}$ produces a sequence of integers, and attempts to recover p by utilizing $\mathcal{A}$ to learn the least-significant bit of the quotients of these integers with respect to p. For this, $\mathcal{B}$ uses the following subroutine:

Subroutine Learn-LSB(z, pk):

Input: $z \in [0, 2^\gamma)$ with $|r_p(z)| < 2^\rho$, a public key $\mathrm{pk} = \langle x_0, x_1, \ldots, x_\tau \rangle$
Output: The least-significant-bit of $q_p(z)$

1. For $j = 1$ to $\mathrm{poly}(\lambda)/\epsilon$ do: // ϵ is the overall advantage of $\mathcal{A}$
2. Choose noise $r_j \xleftarrow{\$} (-2^{\rho'}, 2^{\rho'})$, a bit $m_j \xleftarrow{\$} \{0,1\}$, and a random subset $S_j \subseteq_R \{1, \ldots, \tau\}$
3. Set $c_j \leftarrow \left[z + m_j + 2r_j + 2\sum_{k \in S_j} x_k \right]_{x_0}$
4. Call $\mathcal{A}$ to get a prediction $a_j \leftarrow \mathcal{A}(\mathrm{pk}, c_j)$
5. Set $b_j \leftarrow a_j \oplus \mathsf{parity}(z) \oplus m_j$ // b_j should be the parity of $q_p(z)$
6. Output the majority vote among the b_j's.

In Lemma 4.3 we show that for all but a negligible fraction of the public keys generated by the scheme, the "ciphertext" c_j in line 3 is distributed almost identically to a valid encryption of the bit $[r_p(z)]_2 \oplus m_j$. Note also that since p is odd, we always have $[q_p(z)]_2 = [r_p(z)]_2 \oplus \mathsf{parity}(z)$. It follows that if $\mathcal{A}$ has a noticeable advantage in guessing the encrypted bit under pk (and the conditions in Lemma 4.3 are met), then Learn-LSB(z, pk) will return $[q_p(z)]_2$ with overwhelming probability.

Step 3: Binary GCD. Once we turned $\mathcal{A}$ into an oracle for the least-significant-bit of $q_p(z)$, recovering p is rather straightforward. Perhaps the simplest way of doing it is using the binary GCD algorithm [14]: Given any two integers $z_1 = q_p(z_1) \cdot p + r_p(z_1)$ and $z_2 = q_p(z_2) \cdot p + r_p(z_2)$ (with $r_p(z_i) \ll p$), repeatedly apply the following process to them:

1. If $z_2 > z_1$ then swap them, $z_1 \leftrightarrow z_2$.

2. Use the oracle to learn the parity bit of both $q_p(z_1)$ and $q_p(z_2)$, denote $b_i = [q_p(z_i)]_2$.
3. If both $q_p(z_i)$ are odd then replace z_1 by $z_1 \leftarrow z_1 - z_2$ and set $b_1 \leftarrow 0$.
4. For each z_i with $b_i = 0$, replace z_i by $z_i \leftarrow (z_i - \mathsf{parity}(z_i))/2$. (Note that $z_i - \mathsf{parity}(z_i)$ is even, so the new z_i is an integer.)

Observe that when $p \gg r_p(z_i)$, subtracting the parity bit does not change the quotient with respect to p, only the remainder. That is, $q_p(z_i - \mathsf{parity}(z_i)) = q_p(z_i)$. It follows that when we set $z_i' \leftarrow (z_i - \mathsf{parity}(z_i))/2$ in line 4 (where we know that $q_p(z_i)$ is even), we get

$$q_p(z_i') = q_p(z_i)/2 \quad \text{and} \quad r_p(z_i') = \big(r_p(z_i) - \mathsf{parity}(z_i)\big)/2.$$

We now show that the noise in z_1, z_2 never grows too large in this process. Clearly, setting $z_i' \leftarrow (z_i - \mathsf{parity}(z_i))/2$ in line 4 we have $|r_p(z_i')| \leq (|r_p(z_i)| + 1)/2 \leq |r_p(z_i)|$. Moreover, when we replace z_1 by $z_1' \leftarrow z_1 - z_2$ in line 3 and then by $z_1'' \leftarrow (z_1' - \mathsf{parity}(z_1'))/2$ in line 4, we have

$$|r_p(z_1'')| \;=\; (|r_p(z_1') - \mathsf{parity}(z_1')|)/2 \;=\; \big(|r_p(z_1) - r_p(z_2) - \mathsf{parity}(z_1')|\big)/2 \;\leq\; \max\big\{|r_p(z_1)|, |r_p(z_2)|\big\}$$

Hence the $r_p(z_i)$'s never grow beyond the largest of the initial two, so we always have $p \gg r_p(z_i)$.

This implies that the operations above correspond to the usual operations of the binary GCD algorithm, applied to the $q_p(z_i)$'s. Hence after $O(\gamma)$ iterations we will finally get two integers z_1', z_2' with $z_2' = 0$ and $q_p(z_1')$ being the odd part of $GCD(q_p(z_1), q_p(z_2))$ (for the two initial integers).

Step 4: Recovering p. To recover p, the solver $\mathcal{B}$ draws a pair of elements $z_1^*, z_2^* \stackrel{\$}{\leftarrow} \mathcal{D}_{\gamma,\rho}(p)$ and applies the binary-GCD algorithm to them. With probability at least $\pi^2/6 \approx 0.6$, the odd part of $GCD(q_p(z_1^*), q_p(z_2^*))$ is one, which means that the procedure will output an element $\tilde{z} = 1 \cdot p + r$ with $|r| \leq 2^\rho$. (If this does not happen then $\mathcal{B}$ draws two new integers and tries again.)

Lastly $\mathcal{B}$ repeats the binary-GCD procedure from above using $z_1 = z_1^*$ and $z_2 = \tilde{z}$, and the sequence of parity bits of the $q_p(z_1)$'s in all the iterations spell out the binary representation of $q_p(z_1^*)$. Now $\mathcal{B}$ recovers $p = \lfloor z_1^*/q_p(z_1^*) \rceil$.

Summary. We have shown that $\mathcal{B}$ can recover p given access to a reliable oracle for computing $[q_p(z)]_2$ (for z's with noise much smaller than p). It is left to analyze the probability (over $\mathcal{B}$'s choice of public key) with which the procedure $\mathsf{Learn\text{-}LSB}(z, \mathrm{pk})$ from above is indeed such a reliable oracle.

4.1.1 The Success Probability of $\mathcal{B}$

Below we prove a simple technical lemma about the distribution of ciphertexts in our scheme. Recall that conditioned on some probability-$\frac{1}{2}$ event in our reduction (i.e., $q_p(x_0)$ is odd), the distribution of the public key that $\mathcal{B}$ generates is identical to the correct distribution from the scheme. Let us denote this probability-$\frac{1}{2}$ "good event" by $\mathcal{G}$. In Lemma 4.3 we prove that for every secret key p and for all but a negligible fraction of the public keys (as generated by KeyGen for the secret key p), the procedure that $\mathcal{B}$ uses to generate ciphertexts in line 3 of the subroutine $\mathsf{Learn\text{-}LSB}$ produces a distribution which is statistically close to the ciphertext distribution of the scheme. This lets us analyze the success probability of $\mathcal{B}$, as follows: Let $\mathcal{P}$ be the set of odd integers in $[2^{\eta-1}, 2^\eta)$ for which $\mathcal{A}$ has more than $\varepsilon/2$ advantage

$$\mathcal{P} \stackrel{\text{def}}{=} \big\{p \in [2^{\eta-1}, 2^\eta) \;:\; \mathsf{advantage}(\mathcal{A}) \text{ conditioned on sk} = p \text{ is at least } \varepsilon/2\big\}$$

A counting argument shows that the fraction of odd integers from $[2^{\eta-1}, 2^{\eta})$ that are in $\mathcal{P}$ is at least $\varepsilon/2$. For a given $p \in \mathcal{P}$, we similarly denote by $\mathcal{PK}_p$ the set of public keys for which $\mathcal{A}$ has advantage at least $\varepsilon/4$:

$$\mathcal{PK}_p \stackrel{\text{def}}{=} \{\text{pk for } p \ :\ \mathsf{advantage}(\mathcal{A}) \text{ conditioned on pk is at least } \varepsilon/4\}$$

Again, for every $p \in \mathcal{P}$, the KeyGen algorithm (when using the secret key $\text{sk} = p$) must output $\text{pk} \in \mathcal{PK}_p$ with probability at least $\varepsilon/4$.

Consider now a single run of $\mathcal{B}$ when it is given access to $\mathcal{D}_{\gamma,\rho}(p)$ for some $p \in \mathcal{P}$. With probability 1/2 the "good event" $\mathcal{G}$ happens, in which case the public key that $\mathcal{B}$ produces is negligibly close to the right distribution. Hence conditioned on $\mathcal{G}$, $\mathcal{B}$ generates some $\text{pk} \in \mathcal{PK}_p$ with probability $\varepsilon' \geq \varepsilon/4 - \mathsf{negl}$. Moreover, by Lemma 4.3, with probability $\varepsilon' - \mathsf{negl}$ not only is the public key in $\mathcal{PK}_p$, but also the ciphertext-generation that $\mathcal{B}$ uses in line 3 of $\mathsf{Learn\text{-}LSB}$ "works" for this public key (meaning that the ciphertexts that it generates are chosen from almost the right distribution). If that happens then $\mathcal{A}$ returns the right answer in line 4 of $\mathsf{Learn\text{-}LSB}$ with probability $\varepsilon/4 - \mathsf{negl}$. As that subroutine calls $\mathcal{A}$ for $\text{poly}(\lambda)/\varepsilon$ times and takes majority vote, it will return the right answer with overwhelming probability, and $\mathcal{B}$ will recover the approximate-gcd p.

Thus, when the hidden secret is $p \in \mathcal{P}$ then $\mathcal{B}$ has probability at least $1/2 \cdot (\varepsilon/4 - \mathsf{negl})$ of recovering it in a single run. Repeating the algorithm $\mathcal{B}$ for $(8/\epsilon) \cdot \omega(\log \lambda)$ times will therefore recover such p's with overwhelming probability. Hence we have a solver of complexity $\text{poly}(\lambda, 1/\varepsilon)$ that works with overwhelming probability for every $p \in \mathcal{P}$, so the overall success probability of this solver is at least the density of $\mathcal{P}$, which is at least $\varepsilon/2$. This completes the proof of Theorem 4.2. $\square$

Lemma 4.3. *Fix the parameters* $(\rho, \rho', \eta, \gamma, \tau)$*, fix any* $\text{sk} = p$*, and let* $\text{pk} = \langle x_0, \ldots, x_\tau \rangle$ *be chosen at random as in the* KeyGen *of our scheme. For* every integer $x^* \in [0, 2^\gamma]$ *which is at most* 2^ρ *away from a multiple of* p*, consider the following distribution*

$$\mathcal{C}_{\text{pk}}(x^*) \ = \ \left\{ S \subseteq_R \{1, \ldots, \tau\},\ r \stackrel{\$}{\leftarrow} (-2^{\rho'},\ 2^{\rho'}) \ :\ \mathsf{output}\ c' \leftarrow \left[x^* + 2r + \sum_{i \in S} x_i \right]_{x_0} \right\}$$

Then with overwhelming probability (over the choice of (sk, pk)*),* every distribution $\mathcal{C}_{\text{pk}}(x^*)$ *is statistically close to to the distribution* $\mathsf{Encrypt}(\text{pk}, m = [x^*]_2)$ *(up to a negligible statistical distance).*

Proof. (sketch) Writing $c' = q'p + 2r' + m$, again the argument boils to separate arguments that q' and r' are distributed as in the scheme.

Regarding q', we claim that in the scheme the value $q_p(c)$ of a ciphertext is uniform in $(-q_0/2, q_0/2]$ by the leftover hash lemma; since the summation used to generate c' is as in the scheme, this implies that the value q' is also uniform in $(-q_0/2, q_0/2]$. This claim follows from Lemma 4.4 below. To apply the lemma meaningfully – i.e., to ensure that the distribution is negligibly within uniform – we need $\tau > \log q_0 + \omega(\log \lambda)$, as indicated in our parameter choices.

Regarding the noise r', recall that we choose additional noise for the ciphertext in both the scheme and the reduction, drawn from a distribution with magnitude much larger than the noise of the public-key elements (by a super-polynomial factor). This added noise statistically drowns whatever differences exist in the noise distribution due to the public key elements and the integer x^*. $\square$

Lemma 4.4. *Consider the following distribution. Set* $x_1, \ldots, x_T \xleftarrow{R} \mathbb{Z}_M$ *uniformly and independently, set* $\mathbf{s} \xleftarrow{R} \{0,1\}^T$*, and set* $x_{T+1} \leftarrow \sum_{i=1}^{T} s_i \cdot x_i \bmod M$*. Then,* $(x_1, \ldots, x_T, x_{T+1})$ *is* $\frac{1}{2}\sqrt{M/2^T}$*-uniform over* $\mathbb{Z}_M^{T+1}$*.*

Proof. Define a family $\mathcal{H}$ of hash functions from $\{0,1\}^T$ to $\mathbb{Z}_M$ as follows. The members $h \in \mathcal{H}$ are associated to elements $(h_1, \ldots, h_T) \in \mathbb{Z}_M^T$. For $\mathbf{s} \in \{0,1\}^T$, $h(\mathbf{s})$ is given by $\sum_{i=1}^{T} s_i \cdot h_i \in \mathbb{Z}_M$. This family is clearly 2-universal. Therefore, by the leftover hash lemma (Lemma 2.1), $(h, h(x))$ is $\frac{1}{2}\sqrt{M/2^T}$-uniform over $\mathbb{Z}_M^{T+1}$. □

Note that Lemma 4.4 means in particular that for all but a $\sqrt[4]{2^T/M}$ fraction of the choices of $x_1, \ldots, x_T$, the induced distribution over x_{T+1} is at most $\sqrt[4]{2^T/M}$ away from uniform.

Remark 4.5. We point out that the reduction above (and therefore our scheme) can work with distributions other than the uniform for the q_i's and r_i's. For the r_i's, all we need is that the distribution is "smooth enough" so that taking any fixed number x and adding to it noise r with variance $\gg x^2$ yields a distribution that is almost identical to the distribution of r itself. (In particular we can use a Gaussian distribution for the r_i's, as is done in Regev's scheme [22].)

For the q_i's all we need is that the distribution has high enough entropy to use the leftover hash lemma.

5 Known Attacks

Consider the approximate-gcd instance $\{x_0, \ldots, x_t\}$ where $x_i = pq_i + r_i$. In this section, we first review known attacks on the approximate-gcd problem for two numbers (i.e., when $t = 1$) – including brute-forcing the remainders, continued fractions, and Howgrave-Graham's approximate gcd algorithm [11]. Later, we consider attacks for arbitrarily large values of t – including lattice-based algorithms for simultaneous Diophantine approximation [15], Nguyen and Stern's orthogonal lattice [20], and extensions of Coppersmith's method to multivariate polynomials [5].

5.1 The Approximate GCD of Two Numbers

A simple brute-force attack is to try to guess r_1 and r_2, and verify the guess with a gcd computation. Specifically, for $r_1', r_2' \in (-2^\rho,\ 2^\rho)$, set

$$x_1' \leftarrow x_1 - r_1' \ , \quad x_2' \leftarrow x_2 - r_2' \ , \quad p' \leftarrow \gcd(x_1', x_2')$$

If p' has η bits, output p' as a possible solution. The solution p will definitely be found by this technique, and for our parameter choices, where ρ is much smaller than η, the solution is likely to be unique. The running time of the attack is approximately $2^{2\rho}$.

A variant of the brute-force attack is to set x_1' as above, factor x_1', and, if there is an η-bit factor p', see whether p' is an approximate divisor of x_2'. Since in our parameters γ is substantially greater than η, the attack should use a factoring algorithm whose performance depends primarily on the size of the target factor rather than the size of the entire number being factored. For example, Lenstra's elliptic curve factoring algorithm [16] runs in time roughly $\exp(O(\sqrt{\eta}))$ (with only polynomial dependence on γ), thus resulting in overall attack complexity $\approx 2^{\rho+\sqrt{\eta}}$. The attack time is less if the approximate gcd is known to be smooth, but still exponential in ρ.

Continued fractions seem like a natural way to recover p from x_1 and x_2. Using continued fractions, one obtains a sequence of integer pairs (a_i, b_i) such that $|x_1/x_2 - a_i/b_i| < 1/b_i^2$. Moreover, every pair (s,t) such that $|x_1/x_2 - s/t| < 1/2t^2$ is in the sequence. Since q_1/q_2 is a good approximation of x_1/x_2, one may hope that it occurs as a pair in the sequence; if so, one recovers $p = \lfloor x_1/q_1 \rceil$. However, in our scheme, $|x_1/x_2 - q_1/q_2|$ is not small enough to be recovered using continued fractions. Specifically, we have

$$\left|\frac{x_1}{x_2} - \frac{q_1}{q_2}\right| = \left|\frac{q_2 r_1 - q_1 r_2}{q_2(pq_2 + r_2)}\right| \approx \left|\frac{q_2 r_1 - q_1 r_2}{p}\right| \cdot \frac{1}{q_2^2}$$

where $(q_2 r_1 - q_1 r_2)/p$ in the final term is likely to be *much* larger than 1. To describe the failure of continued fractions another way, the mere fact that an approximant a_i/b_i is close to x_1/x_2 does *not* mean that there exist $r_1', r_2' \ll p$ such that $x_1 = pa_i + r_1'$ and $x_2 = pb_i + r_2'$ – i.e., the continued fractions method is not constrained to output the kind of approximants that we need. See [11] for a more detailed exposition of the continued fractions approach to approximate-gcd.

Howgrave-Graham [11] also gives a lattice attack on the two-element approximate-gcd problem that is related to Coppersmith's celebrated algorithm for finding small solutions to univariate and bivariate modular equations [5]. For the case where x_1 is exactly divisible by p, where his algorithm performs slightly better, the attack recovers p when ρ/γ is smaller than $(\eta/\gamma)^2$. The algorithm does not degrade gracefully for ρ, η, γ that do not satisfy the constraint. Rather, in this case, the relevant lattice may contain exponentially vectors unrelated to the approximate-gcd solution, so that lattice reduction yields nothing useful.

5.2 The Approximate GCD of Many Numbers

Now, let us consider attacks – specifically, lattice attacks – for arbitrary t. First, note that the rational numbers $y_i = x_i/x_0$ are an instance of the simultaneous Diophantine approximation (SDA) problem: indeed for all i it holds that $\frac{x_i}{x_0} = \frac{q_i + s_i}{q_0}$, where $|s_i| \approx 2^{\rho-\eta}$. We can therefore try to use Lagarias' algorithm for SDA [15], namely apply LLL to the $(t+1)$-dimensional lattice L spanned by the rows of the following matrix:

$$M = \begin{pmatrix} 2^\rho & x_1 & x_2 & \dots & x_t \\ & -x_0 & & & \\ & & -x_0 & & \\ & & & \ddots & \\ & & & & -x_0 \end{pmatrix}$$

Our target solution corresponds to a vector of length roughly $2^{\gamma+\rho-\eta}\sqrt{t+1}$ – specifically,

$$\begin{aligned} \vec{v} = \langle q_0, q_1, \dots, q_t \rangle \cdot M &= \langle q_0 2^\rho,\ q_0 x_1 - q_1 x_0,\ \dots,\ q_0 x_t - q_t x_0 \rangle \\ &= \left\langle q_0 2^\rho,\ x_0 q_0 (\frac{x_1}{x_0} - \frac{q_1}{q_0}),\ \dots,\ x_0 q_0 (\frac{x_t}{x_0} - \frac{q_t}{q_0}) \right\rangle, \end{aligned}$$

where the first entry in $\vec{v}$ satisfies $|q_0 2^\rho| < 2^{\gamma-\eta+\rho}$ and all the other entries satisfy $|x_0 q_0 (\frac{x_i}{x_0} - \frac{q_i}{q_0})| = |x_0 s_i| \approx 2^{\gamma+\rho-\eta}$.

However, the target solution is not necessarily the shortest nonzero vector in the lattice, and therefore is not necessarily discovered by lattice reduction. In particular, Minkowski tells us that L has a nonzero vector of length at most $\det(L)^{1/(t+1)}\sqrt{t+1} < 2^{(\rho+t\gamma)/(t+1)}\sqrt{t+1} = 2^{\gamma+(\rho-\gamma)/(t+1)}\sqrt{t+1}$. This is shorter than our target solution when $t+1 < \gamma/\eta$. In fact, heuristically, L will tend to have exponentially (in t) many vectors of length $\mathrm{poly}(t)\det(L)^{1/(t+1)}$, which obscure our target solution.[5]

On the other hand, when t is large, $\vec{v}$ likely is the shortest vector in L, but known lattice reductions algorithms will not be able to find it efficiently. Specifically, as a rule of thumb, they require time roughly $2^{t/k}$ to output a 2^k approximation of the shortest vector. Since clearly there are exponentially (in t) many vectors in L of length at most $\|x_0\|\sqrt{t+1} < 2^\gamma\sqrt{t+1}$, which is about $2^{\eta-\rho}$ times longer than $\vec{v}$, we need better than a $2^{\eta-\rho}$ approximation. For $t \geq \gamma/\eta$, the time needed to guarantee a 2^η approximation (which is not even good enough to recover $\vec{v}$) is roughly $2^{\gamma/\eta^2}$. Thus setting $\gamma/\eta^2 = \omega(\log\lambda)$ foils this attack.

Other known attacks are described in Appendix B. These attacks do not perform any better than the ones above, and our choice of parameters achieves at least 2^λ security against all of them.

6 Making the Scheme Fully Homomorphic

We follow Gentry's approach [7] for constructing a fully homomorphic encryption scheme from a somewhat homomorphic scheme $\mathcal{E}$ that is *bootstrappable* as per Definition 2.6. For reasons similar to those in Gentry's construction from [7], computing the decryption equation $m' \leftarrow [c - \lfloor c/p \rceil]_2$ seems to require boolean circuits that are deeper (by a constant factor) than what our somewhat homomorphic scheme can handle. Hence we use Gentry's transformation to "squash the decryption circuit." In this transformation, we add to the public key some extra information about the secret key, and use this extra information to "post process" the ciphertext. The post-processed ciphertext can be decrypted more efficiently than the original ciphertext, thus making the scheme bootstrappable. We pay for this saving by having a larger ciphertext, and also by introducing another hardness assumption (basically assuming that the extra information in the public key does not help an attacker break the scheme).

6.1 Squashing the Decryption Circuit

Let κ, θ, Θ be three more parameters, which are functions of λ. Concretely, below we use $\kappa = \gamma\eta/\rho'$, $\theta = \lambda$, and $\Theta = \omega(\kappa \cdot \log\lambda)$.[6] For a secret key $\mathrm{sk}^* = p$ and public key pk^* from the original somewhat homomorphic scheme $\mathcal{E}^*$, we add to the public key a set $\vec{y} = \{y_1, \ldots, y_\Theta\}$ of rational numbers in $[0, 2)$ with κ bits of precision, such that there is a sparse subset $S \subset \{1, \ldots, \Theta\}$ of size θ with $\sum_{i \in S} y_i \approx 1/p \pmod 2$. We also replace the secret key by the indicator vector of the subset S. In more details, we modify the encryption scheme from Section 3 as follows:

KeyGen. Generate $\mathrm{sk}^* = p$ and pk^* as before. Set $x_p \leftarrow \lfloor 2^\kappa/p \rceil$, choose at random a Θ-bit vector with Hamming weight θ, $\vec{s} = \langle s_1, \ldots, s_\Theta \rangle$, and let $S = \{i : s_i = 1\}$.

Choose at random integers $u_i \in \mathbb{Z} \cap [0, 2^{\kappa+1})$, $i = 1, \ldots, \Theta$, subject to the condition that $\sum_{i \in S} u_i = x_p \pmod{2^{\kappa+1}}$. Set $y_i = u_i/2^\kappa$ and $\vec{y} = \{y_1, \ldots, y_\Theta\}$. Hence each y_i is a positive

[5] When t is very small – e.g., $t = 1$ – the information that one obtains from the two dimensional lattice is related to what one obtains from the continued fractions approach.

[6] When using the size-reduction optimization from Section 3.3 it is sufficient to use $\kappa = \gamma + 2$, which would also make Θ smaller.

number smaller than two, with κ bits of precision after the binary point. Also, $[\sum_{i\in S} y_i]_2 = (1/p) - \Delta_p$ for some $|\Delta_p| < 2^{-\kappa}$.

Output the secret key $\mathrm{sk} = \vec{s}$ and public key $\mathrm{pk} = (\mathrm{pk}^*, \vec{y})$.

Encrypt and Evaluate. Generate a ciphertext c^* as before (i.e., an integer). Then for $i \in \{1, \ldots, \Theta\}$, set $z_i \leftarrow [c^* \cdot y_i]_2$, keeping only $n = \lceil \log \theta \rceil + 3$ bits of precision after the binary point for each z_i. Output both c^* and $\vec{z} = \langle z_1, \ldots, z_\Theta \rangle$.

Decrypt. Output $m' \leftarrow \left[c^* - \lfloor \sum_i s_i z_i \rceil\right]_2$.

Recall our definition of permitted polynomials from Section 3.2. We proved that our somewhat homomorphic scheme was correct for the set $C(\mathcal{P}_\mathcal{E})$ of circuit that compute permitted polynomials, and we now show that this is true also of the modified scheme.

Lemma 6.1. *The modified scheme from above is correct for $C(\mathcal{P}_\mathcal{E})$. Moreover, for every ciphertext $(c^*, \vec{z})$ that is generated by evaluating a permitted polynomial, it holds that $\sum s_i z_i$ is within 1/4 of an integer.*

Proof. Fix public and secret keys, generated with respect to security parameter λ, with $\{y_i\}_{i=1}^{\Theta}$ the rational numbers in the public key and $\{s_i\}_{i=1}^{\Theta}$ the secret-key bits. Recall that the y_i's were chosen so that $[\sum_i s_i y_i]_2 = (1/p) - \Delta_p$ with $|\Delta_p| \leq 2^{-\kappa}$.

Fix a permitted polynomial $P(x_1, \ldots, x_t) \in \mathcal{P}_\mathcal{E}$, an arithmetic circuit C that computes P, and t ciphertexts $\{c_i\}_{i=1}^{t}$ that encrypt the inputs to C, and denote $c^* = \mathsf{Evaluate}(\mathrm{pk}, C, c_1, \ldots, c_t)$. We need to establish that

$$\lfloor c^*/p \rceil = \left\lfloor \sum_i s_i z_i \right\rceil \pmod 2$$

where the z_i's are computed as $[c^* \cdot y_i]_2$ with only $\lceil \log \theta \rceil + 3$ bits of precision after the binary point, so $[c^* \cdot y_i]_2 = z_i - \Delta_i$ with $|\Delta_i| \leq 1/16\theta$. We have

$$\begin{aligned}
\left[(c^*/p) - \sum s_i z_i\right]_2 &= \left[(c^*/p) - \sum s_i [c^* \cdot y_i]_2 + \sum s_i \Delta_i\right]_2 \\
&= \left[(c^*/p) - c^* \cdot \left[\sum s_i y_i\right]_2 + \sum s_i \Delta_i\right]_2 \\
&= \left[(c^*/p) - c^* \cdot (1/p - \Delta_p) + \sum s_i \Delta_i\right]_2 \\
&= \left[c^* \cdot \Delta_p + \sum s_i \Delta_i\right]_2
\end{aligned}$$

We claim that the final quantity inside the brackets has magnitude at most 1/8. By definition, since c^* is a valid ciphertext output by a permitted polynomial, the value c^*/p is within 1/8 of an integer. Together, these facts imply the lemma.

To establish the claim, observe that $|\sum s_i \Delta_i| \leq \theta \cdot \frac{1}{16\theta} = 1/16$. Regarding $c^* \cdot \Delta_p$, recall that the output ciphertext c^* is obtained by evaluating the polynomial P on the input ciphertexts c_i (as if P was an integer polynomial). By the definition of a permitted polynomial, for any $\alpha \geq 1$, if P's inputs have magnitude at most $2^{\alpha(\rho'+2)}$, its output has magnitude at most $2^{\alpha(\eta-4)}$ when its inputs have magnitude. In particular, when P's inputs are "fresh" ciphertexts, which have magnitude at most 2^γ, P's output ciphertext c^* has magnitude at most $2^{\gamma(\eta-4)/(\rho'+2)} < 2^{\kappa-4}$. Thus, $|c^* \cdot \Delta_p| < 1/16$ and the claim follows.

□

6.2 Bootstrapping Achieved!

Theorem 6.2. *Let $\mathcal{E}$ be the scheme above, and let $D_{\mathcal{E}}$ be the set of augmented (squashed) decryption circuits. Then, $D_{\mathcal{E}} \subset C(\mathcal{P}_{\mathcal{E}})$.*

In other words, $\mathcal{E}$ is bootstrappable. The proof is similar to Gentry's [6, 7]. By Theorem 2.7, we obtain homomorphic encryption schemes for circuits of any depth.

Proof. The goal is to express the modified decryption equation

$$m' \leftarrow c^* - \left\lfloor \sum s_i \cdot z_i \right\rceil \bmod 2$$

as a permitted polynomial (i.e., one satisfying Equation (1)), and show that there is a polynomial-size circuit that computes this polynomial. Recall that c^* is an integer, the s_i's are bits, and the z_i's are rational numbers in $[0, 2)$, in binary representation with $n = \lceil \log \theta \rceil + 3$ bits of precision after the binary point. Also, our parameter setting implies two promises – namely, that $\sum s_i \cdot z_i$ is within 1/4 of an integer, and that only θ of the bits $s_1, \ldots, s_\Theta$ are nonzero.

We split the computation up into three steps:

1. For $i \in \{1, \ldots, \Theta\}$, set $a_i \leftarrow s_i \cdot z_i$ (i.e., $a_i = z_i$ when $s_i = 1$ and $a_i = 0$ otherwise). The a_i's are still rational numbers in $[0, 2)$, given in binary representation with n bits of precision after the binary point.

2. From the Θ rational numbers $\{a_i\}_{i=1}^{\Theta}$, generate other $n+1$ rational numbers $\{w_j\}_{j=0}^{n}$, each with less than n bits of precision, such that $\sum_j w_j = \sum_i a_i \pmod 2$.

3. Output $c^* - (\sum_j w_j) \bmod 2$.

The first step can be performed with a 1-level sub-circuit of multiplication gates. However, the second and third steps require more complicated sub-circuits.

The problem of using a shallow boolean circuit to compute the sum $\sum_{i=1}^{k} r_i$ of k rational numbers in binary representation is well-studied. A well-known technique uses the three-for-two trick (see [13]), whereby a constant-depth circuit is used to transform three numbers of arbitrary bit-length into two numbers that are at most 1 bit longer, such that the sum of the two output numbers is the same as the sum of the three input numbers. (The output bits of the constant-depth circuit are linear or quadratic expressions with 3 monomials in the input bits.) By applying this trick at most $\left\lceil \log_{3/2} k \right\rceil + 2$ times, one obtains two numbers s_1 and s_2 such that $s_1 + s_2 = \sum_{i=1}^{k} r_i$. Hence the total depth that it takes to reduce k numbers to two numbers is $d' \leq 2^{\left\lceil \log_{3/2} k \right\rceil + 2} < 8k^{1/\log(3/2)} < 8k^{1.71}$. The depth of the circuit needed to compute the final sum of two numbers is logarithmic in their bit-lengths, but if we are only interested in $\lfloor s_1 + s_2 \rceil \bmod 2$ and have the promise that $s_1 + s_2$ is within 1/4 of an integer, this value can be computed by multivariate polynomial of degree 4 (and only nine terms). Overall, the circuit for computing $\left\lfloor \sum_{i=1}^{k} r_i \right\rceil \bmod 2$ corresponds to a polynomial of degree at most $d \leq 32k^{1/\log(3/2)}$. with coefficient vector having l_1-norm at most 27^d. Unfortunately, this degree (with $k = \Theta$) is still too large for our scheme to handle. Hence we use Gentry's technique from [6] that takes advantage of the fact that all but θ of the a_i's are zero.

Denote the bit representation of each number a_i by $a_{i,0} {\bullet} a_{i,-1} a_{i,-2} \ldots a_{i,-n}$. That is, $a_i = \sum_{j=0}^{n} 2^{-j} a_{i,-j}$. The heart of this procedure is a subroutine for computing integers W_{-j}, $j =$

$$
\begin{array}{ccccc}
a_{1,0} & a_{1,-1} & a_{1,-2} & & a_{1,-n} \\
a_{2,0} & a_{2,-1} & a_{2,-2} & & a_{2,-n} \\
a_{3,0} & a_{3,-1} & a_{3,-2} & & a_{3,-n} \\
 & & & \cdots & \\
a_{\Theta,0} & a_{\Theta,-1} & a_{\Theta,-2} & & a_{\Theta,-n} \\
\hline
W_0 & W_{-1} & W_{-2} & & W_{-n}
\end{array}
$$

Figure 1: The procedure for summing up the a_i's: The binary representation of the rational number a_i is $a_{i,0} \bullet a_{i,-1} a_{i,-2} \ldots a_{i,-n}$. The integer W_{-j} is the Hamming weight of the column of bits $(a_{1,-j}, a_{2,-j}, \ldots, a_{\Theta,-j})$.

$0, 1, \ldots, n$, where W_{-j} is the Hamming weight of the "column" of bits $(a_{1,-j}, a_{2,-j}, \ldots, a_{\Theta,-j})$ (see an illustration in Figure 1). Since at most θ of the a_i's are nonzero, then the W_{-j}'s are no larger than θ, and hence can be represented by $\lceil \log(\theta+1) \rceil < n$ bits. By Lemma 6.3 below, every bit in the binary representation of W_{-j} can be expressed as a polynomial of degree at most θ in the Θ variables $a_{i,-j}$, $i = 1, 2, \ldots, \Theta$. Moreover all of these polynomials can be computed simultaneously by an arithmetic circuit of size $O(\theta \cdot \Theta)$.

Once we have the W_{-j}'s, the sum of the a_i's can be obtained by $\sum_i a_i = \sum_j 2^{-j} W_{-j}$. For $j = 0, 1, \ldots, n$ we set $w_j = (2^{-j} \cdot W_{-j}) \bmod 2$, so the w_j's are rational numbers with $\lceil \log(\theta+1) \rceil < n$ bits of precision. We can now sum-up the w_j's using the three-for-two trick as above, this time with $k = n+1$, thus obtaining the sum of the a_i's mod 2.

We conclude that the degree of the polynomials in the first step is two, the degree of polynomials in the second step is at most θ, and the degree of the polynomial in the third step is at most

$$32(n+1)^{1/\log(3/2)} \;<\; 32\,\lceil \log\theta + 4 \rceil^{1.71} \;<\; 32 \log^2\theta$$

Therefore the total degree of the decryption circuit is bounded by $2 \cdot \theta \cdot 32 \log^2\theta = 64\theta \log^2\theta$, and since we are using $\theta = \lambda$ we have degree at most $64\lambda \log^2\lambda$.

It follows that the augmented decryption circuits $D_{\mathcal{E}}$ (i.e., decryption followed by a single multiplication or addition, cf. Definition 2.5) can be expressed as polynomials of degree at most $128\lambda \log^2\lambda$ in the Θ variables s_i. Since the logarithm of l_1-norm of this polynomial is small in relation to η, and since $\Theta = \frac{\eta\gamma}{\rho} \cdot \omega(\log\lambda) < \lambda^7$ (and also $\tau < \lambda^7$) the argument in Remark 3.6 at the end of Section 3.2 (with $\alpha = 128$ and $\beta = 7$) indicates that we can get $D_{\mathcal{E}} \subset C(\mathcal{P}_{\mathcal{E}})$, making the scheme bootstrappable, by setting $\eta = \rho \cdot \Theta(\lambda \log^2\lambda)$. $\square$

It is left to show how to compute the W_j's using polynomials of degree no larger than θ.

Lemma 6.3. *Let $\vec{\sigma} = \langle \sigma_1, \sigma_2, \ldots, \sigma_t \rangle$ be a binary vector, let $W = W(\vec{\sigma})$ be the Hamming weight of $\vec{\sigma}$, and denote the binary representation of W by $W_n \ldots W_1 W_0$. (That is, $W = \sum_{i=0}^{n} 2^i W_i$ and all the W_i's are bits.)*

Then for every $i \le n$, the bit $W_i(\vec{\sigma})$ can be expressed as a binary polynomial of degree exactly 2^i in the variables $\sigma_1, \ldots, \sigma_t$. Moreover, there is an arithmetic circuit of size $2^i \cdot t$ that simultaneously computes all the polynomials for $W_0, \ldots, W_i$.

Proof. It is well known that the i'th bit in the binary representation of the Hamming weight of bit-vector $\vec{\sigma}$ is equal to $e_{2^i}(\vec{\sigma})$ modulo 2, where $e_k(\cdot)$ is the k'th elementary symmetric polynomial, see Lemma 4 of [2]. That is,

$$W_i(\vec{\sigma}) \;=\; e_{2^i}(\vec{\sigma}) \bmod 2 \;=\; \left(\sum_{|S|=2^i} \prod_{j\in S} \sigma_j\right) \bmod 2$$

Clearly, the degree of e_{2^i} is exactly 2^i.

As for the "Moreover" part, we can compute the elementary symmetric polynomials in the σ_i's as the coefficients of the polynomial $P_{\vec{\sigma}}(z) = \prod_{i=1}^{t}(z - \sigma_i)$ in the auxiliary formal variable z, with $e_k(\vec{\sigma})$ being the coefficient of z^{t-k}. Conveniently, to compute only the first few bits $W_0, W_1, \ldots, W_i$, we can simply discard the lower-order terms in $P_{\vec{\sigma}}(z)$ – i.e., we do not need the coefficients of z^j for $j < t - 2^i$.

For example, one "dynamic programming" procedure for computing W_0, W_1, …, W_i (which can be trivially made into a circuit) would go as follows:

Input: bits $\sigma_1, \ldots, \sigma_t$
0. Initialization: Set $P_{0,0} \leftarrow 1$ and $P_{j,0} \leftarrow 0$ for $j = 1, 2, 3, \ldots, 2^i$
   // $P_{j,k}$ is the j'th symmetric polynomial in $\sigma_1 \ldots \sigma_k$
1. For $k = 1, 2, \ldots, t$ // incorporate σ_k
2. For $j = 2^i$ down to 1, set $P_{j,k} \leftarrow \sigma_k \times P_{j-1,k-1} + P_{j,k-1}$
3. Output $P_{1,t}, P_{2,t}, P_{4,t}, \ldots, P_{2^i,t}$

We can do a little better by using fast Fourier transform multiplication of polynomials. Using this technique, we can compute the entire polynomial $P_{\vec{\sigma}}(z)$ with complexity $t \cdot \mathrm{polylog}(t)$.

□

Remark 6.4. Note that our first circuit implementation of the procedure from above is not "shallow". Nonetheless, since it computes only "low degree polynomials" (i.e., up to degree 2^i), then by Lemma 3.5 it is a permitted circuit.

6.3 Security of the Squashed Scheme

Putting the hint $\vec{y}$ in the public key induces another computational assumption, related to the sparse subset sum problem (SSSP) used by Gentry [6], and studied previously (sometimes under the name "low-weight" knapsack) in the context of server-aided cryptography [19] and in connection to the Chor-Rivest cryptosystem [21]. We can easily avoid known attacks on the problem by choosing θ large enough to avoid brute-force attacks (and improvements using time-space trade-offs) and choosing Θ to be larger than $\omega(\log\lambda)$ times the bit-length of the rational numbers in the public key (which have length κ).[7]

[7]Note that the SSSP instance and the approximate-GCD instance share the same integer p, but this is not a problem since SSSP is considered hard even if the attacker knows p.

7 Conclusion and Open Problems

We described a fully homomorphic encryption scheme that uses only simple integer arithmetic. The primary open problem is to improve the efficiency of the scheme, to the extent that it is possible while preserving the hardness of the approximate-gcd problem.

References

[1] W. Alexi, B. Chor, O. Goldreich, and C.-P. Schnorr. Rsa and rabin functions: Certain parts are as hard as the whole. *SIAM J. Comput.*, 17(2):194–209, 1988.

[2] J. Boyar, R. Peralta, and D. Pochuev. On the multiplicative complexity of boolean functions over the basis $(\wedge, \oplus, 1)$. *Theor. Comput. Sci.*, 235(1):43–57, 2000.

[3] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In *Advances in Cryptology - EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 402–414. Springer, 1999.

[4] B. Cohen. Web document, `http://bramcohen.com/simple_public_key.html`, 2000. See also `http://www.mail-archive.com/cypherpunks@cyberpass.net/msg00018.html`.

[5] D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *J. Cryptology*, 10(4):233–260, 1997.

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

[7] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC '09*, pages 169–178. ACM, 2009.

[8] C. Gentry and Z. Ramzan. Single-database private information retrieval with constant communication rate. In *ICALP'05*, volume 3580 of *Lecture Notes in Computer Science*, pages 803–815. Springer, 2005.

[9] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.

[10] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

[11] N. Howgrave-Graham. Approximate integer common divisors. In *CaLC '01*, volume 2146 of *Lecture Notes in Computer Science*, pages 51–66. Springer, 2001.

[12] Y. Ishai and A. Paskin. Evaluating branching programs on encrypted data. In *4th Theory of Cryptography Conference (TCC'07)*, volume 4392 of *Lecture Notes in Computer Science*, pages 575–594. Springer, 2007.

[13] R. M. Karp and V. Ramachandran. A Survey of Parallel Algorithms for Shared-Memory Machines. Technical Report CSD-88-408, UC Berkeley, 1988.

[14] D. E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, 3rd edition, 1997.

[15] J. C. Lagarias. The computational complexity of simultaneous diophantine approximation problems. *SIAM J. Comput.*, 14(1):196–209, 1985.

[16] A. K. Lenstra. Factoring multivariate polynomials over algebraic number fields. *SIAM J. Comput.*, 16(3):591–598, 1987.

[17] É. Levieil and D. Naccache. Cryptographic test correction. In R. Cramer, editor, *Public Key Cryptography*, volume 4939 of *Lecture Notes in Computer Science*, pages 85–100. Springer, 2008.

[18] Y. Lindell and B. Pinkas. A proof of security of yao's protocol for two-party computation. *J. Cryptology*, 22(2), 2009.

[19] P. Q. Nguyen and I. Shparlinski. On the insecurity of a server-aided RSA protocol. In *Advances in Cryptology - ASIACRYPT'01*, volume 2248 of *Lecture Notes in Computer Science*, pages 21–35. Springer, 2001.

[20] P. Q. Nguyen and J. Stern. The two faces of lattices in cryptology. In *Cryptography and Lattices, CaLC'01*, volume 2146 of *Lecture Notes in Computer Science*, pages 146–180. Springer, 2001.

[21] P. Q. Nguyen and J. Stern. Adapting density attacks to low-weight knapsacks. In *Advances in Cryptology - ASIACRYPT'05*, volume 3788 of *Lecture Notes in Computer Science*, pages 41–58. Springer, 2005.

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

[23] A. C. Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science – FOCS '82*, pages 160–164. IEEE, 1982.

A Proof of Correctness

Proof. (Lemma 3.3] We first consider a "fresh" ciphertext output by Encrypt. The following lemma says that each such ciphertext is close to a multiple of p, and that its difference from the closest multiple of p has the same parity as m.

Lemma A.1. *Let* $(\mathrm{sk}, \mathrm{pk})$ *be output by* $\mathsf{KeyGen}(\lambda)$. *Let* $c \xleftarrow{\mathrm{R}} \mathsf{Encrypt}(\mathrm{pk}, m)$ *for* $m \in \{0,1\}$. *Then,* $c = a \cdot p + (2b + m)$ *for some integers* a *and* b *with* $|2b + m| \le \tau 2^{\rho+3}$.

Proof. By definition, $c \leftarrow \left[m + 2r + \sum_{i \in S} x_i\right]_{x_0}$. Since $|x_0| \ge |x_i|$ for $i \in \{1, \ldots, \tau\}$, we have that

$$c = \left(m + 2r + \sum_{i \in S} x_i\right) + k \cdot x_0 \quad \text{for some } |k| \le \tau.$$

For every i, there exist integers q_i and r_i with $|r_i| \le 2^\rho$ such that $x_i = q_i \cdot p + 2r_i$, and also $|r_i| \le 2^\rho$. We have

$$c = p \cdot \left(kq_0 + \sum_{i \in S} q_i\right) + \left(m + 2r + k \cdot 2r_0 + \sum_{i \in S} 2r_i\right)$$

Regarding the rightmost term, its parity is the same as m, and its absolute value is at most $(4\tau+3)2^{\rho} < \tau 2^{\rho+3}$, as claimed. $\square$

The following lemma says that essentially the same is true for ciphertexts output by $\mathsf{Evaluate}$ – i.e., ciphertexts are close to multiples of p, though possibly not as close.

Lemma A.2. *Let* $(\mathrm{sk}, \mathrm{pk})$ *be output by* $\mathsf{KeyGen}(\lambda)$. *Let* $C \in \mathcal{C}_{\mathcal{E}}$ *be a circuit with* t *inputs and one output. For* $i \in \{1, \dots, t\}$ *and* $m_i \in \{0,1\}$, *let* $c_i \stackrel{\mathrm{R}}{\leftarrow} \mathsf{Encrypt}(\mathrm{pk}, m_i)$. *Let* $m \leftarrow C(m_1, \dots, m_t)$ *and* $c \leftarrow \mathsf{Evaluate}(\mathrm{pk}, C, c_1, \dots, c_t)$. *Then,* $c = a \cdot p + (2b+m)$ *for some integers* a *and* b *with* $|2b+m| < p/8$.

Proof. Let C' be the generalized circuit corresponding to C, which operates over the integers rather than modulo 2. Generally, we have that $C'(c_1, \dots, c_t) \in C'(2b_1+m_1, \dots, 2b_t+m_t) + p\mathbb{Z}$. So $[C'(2b_1+m_1, \dots, 2b_t+m_t)]_p$ has the same parity as $m = C(m_1, \dots, m_t)$. We also have that $|C'(2b_1+m_1, \dots, 2b_t+m_t)| \le 2^{\eta}/16 \le p/8$ by the definition of $\mathcal{C}_{\mathcal{E}}$, since $|2b_i + m_i| \le \tau 2^{\rho+3}$ by Lemma A.1. $\square$

Lemmas A.1 and A.2 immediately imply Lemma 3.3: for any circuit in $\mathcal{C}_{\mathcal{E}}$ and any encryptions of inputs to that circuit, the integer output by $\mathsf{Evaluate}$ is of the form $c = a \cdot p + (2b+m)$ with $|2b+m| \le p/8$ (where m is the plaintext that c is supposed to encrypt). Accordingly, we have $[c]_p = 2b+m$, and thus $m = \left[[c]_p\right]_2$. $\square$

B More Known Attacks

B.1 Using Nguyen and Stern's Orthogonal Lattice

As another lattice attack, consider the t-dimensional lattice $L_{\vec{x}}^{\perp}$ of integer vectors orthogonal to $\vec{x} = (x_0, \dots, x_t)$. We have $\det(L_{\vec{x}}^{\perp}) = \|\vec{x}\|$. (See Nguyen and Stern's discussion of the orthogonal lattice [20].) Heuristically, if $\vec{x}$ were "random," we would expect the shortest nonzero vector in $L_{\vec{x}}^{\perp}$ to have length about $\sqrt{t}\det(L_{\vec{x}}^{\perp})^{1/t} \approx \sqrt{t} \cdot 2^{\gamma/t}$. However $\vec{x} = p \cdot \vec{q} + \vec{r}$, where $\vec{q} = (q_0, \dots, q_t)$ and $\vec{r} = (r_0, \dots, r_t)$. Therefore, any vector that is orthogonal to both $\vec{q}$ and $\vec{r}$ – i.e., is in the lattice $L_{\vec{q},\vec{r}}^{\perp}$ – is also in $L_{\vec{x}}^{\perp}$. For certain parameters, there will likely be $t-1$ linearly independent vectors in $L_{\vec{q},\vec{r}}^{\perp}$ that are much shorter than any vectors in $L_{\vec{x}}^{\perp} \setminus L_{\vec{q},\vec{r}}^{\perp}$. The idea of the attack is to reduce $L_{\vec{x}}^{\perp}$ to recover these $t-1$ vectors of $L_{\vec{q},\vec{r}}^{\perp}$, from which we can recover $\vec{q}$ and $\vec{r}$, and hence p.

Consider the length of short vectors in $L_{\vec{q},\vec{r}}^{\perp}$. The determinant of $L_{\vec{q},\vec{r}}^{\perp}$ is at most $\|\vec{q}\| \cdot \|\vec{r}\|$. Since $L_{\vec{q},\vec{r}}^{\perp}$ has dimension $t-1$, we expect $L_{\vec{q},\vec{r}}^{\perp}$ to have vectors of length approximately $\sqrt{t}2^{(\gamma+\rho-\eta)/(t-1)}$.

Now, consider the length of the shortest vector in $L_{\vec{x}}^{\perp} \setminus L_{\vec{q},\vec{r}}^{\perp}$. $L_{\vec{x}}^{\perp}$ has one more dimension than $L_{\vec{q},\vec{r}}^{\perp}$, and including this dimension increases the determinant by a factor of $\det(L_{\vec{x}}^{\perp})/\det(L_{\vec{q},\vec{r}}^{\perp}) \approx \|\vec{x}\|/\|\vec{q}\|\|\vec{r}\| \approx 2^{\eta-\rho}$. The shortest vector in $L_{\vec{x}}^{\perp} \setminus L_{\vec{q},\vec{r}}^{\perp}$ therefore has length at most about $2^{\eta-\rho} + (1/2)\sum_{i=1}^{t-1}\|\vec{b}_i\|$, where the $\vec{b}_i$'s are the shortest vectors in $L_{\vec{x}}^{\perp}$.

To recover vectors in $L_{\vec{q},\vec{r}}^{\perp}$ using lattice reduction, the attacker needs the shortest nonzero vector in $L_{\vec{x}}^{\perp} \setminus L_{\vec{q},\vec{r}}^{\perp}$ to be longer than an independent set of vectors in $L_{\vec{q},\vec{r}}^{\perp}$. This occurs roughly when $2^{\eta-\rho} > 2^{(\gamma+\rho-\eta)/(t-1)} \Rightarrow t > \gamma/(\eta-\rho)$.

On the other hand, when $t > \gamma/(\eta-\rho)$ is large, lattice reduction algorithms will not be able to recover the short vectors in $L_{\vec{q},\vec{r}}^{\perp}$ efficiently. Similar to the analysis of the SDA attack, we need our lattice reduction algorithm to provide at least a $2^{\eta-\rho}$ approximation to obtain vectors guaranteed to be in $L^{\perp}_{\vec{q},\vec{r}}$, and (for such large t) known lattice reduction algorithms take time roughly $2^{\gamma/\eta^2}$ to guarantee such a close approximation.

A very similar attack is the following. Consider the lattice spanned by the rows of the following $t \times (t+1)$ matrix, where row i corresponds to the constraint $x_i - r_i = 0 \pmod{p}$, and R_i is an upper bound on $|r_i|$.

$$M = \begin{bmatrix} x_1 & R_1 & & & \\ x_2 & & R_2 & & \\ \vdots & & & \ddots & \\ x_t & & & & R_t \end{bmatrix}$$

Let $\vec{v} = \langle v_0, v_1, \ldots, v_t \rangle$ be a vector in this lattice, namely $\vec{v} = \sum_{i=1}^{t} \mu_i M_i$ for some integers μ_i. Then, we have

$$v_0 - \sum_{i=1}^{t} \frac{v_i}{R_i} \cdot r_i \;=\; \sum_{i=1}^{t} \mu_i x_i - \sum_{i=1}^{t} \frac{\mu_i R_i}{R_i} \cdot r_i \;=\; \sum_{i=1}^{t} \mu_i (x_i - r_i) \;=\; 0 \pmod{p}$$

Suppose that the l_1 norm of $\vec{v}$ is at most $p/2$. Then, $|v_0 - \sum_{i=1}^{t} \frac{v_i}{R_i} \cdot r_i| \leq \sum_{i=0}^{t} |v_i| \leq p/2$. Since this quantity also equals 0 modulo p, we have that $v_0 - \sum_{i=1}^{t} \frac{v_i}{R_i} \cdot r_i = 0$ *over the integers*. If we heuristically suppose that we can find t such $\vec{v}$'s, all of which are orthogonal to $(1, -\frac{r_1}{R_1}, \ldots, -\frac{r_t}{R_t})$, then we can solve for the r_i's.

The determinant of this lattice is bounded by the product of the norms of the columns of M, namely at most $\sqrt{t} X R^t$, where X is an upper bound on $|x_i|$. Hence it has vectors roughly as short as $R\sqrt[t]{X}$. To obtain suitable vectors $\vec{v}$, we need this quantity to be at most p. We obtain $R\sqrt[t]{X} < p$ when $t > \gamma/(\eta - \rho)$. The rest of the analysis is similar to that for the previous attack.

B.2 Extending Coppersmith's Method

We may attempt to use Coppersmith's technique [5] to augment the last attack from above. Namely, instead of looking only at the relations $x_i - r_i = 0 \pmod{p}$, we can also look at their products, e.g. $(x_i - r_i)^2 = 0 \bmod p^2$, $(x_i - r_i)(x_j - r_j) = 0 \bmod p^2$, etc. As an illustrative example, consider the case of products of up to two relations from a total of $t = 3$ available relations. We then have a matrix whose rows correspond to the $\binom{t+1}{2} = 6$ possible pairs of relations, and whose columns correspond to the $\binom{t+2}{2} = 10$ terms that appear in these product relations:

$$M = \left(\begin{array}{cccccccccc} 1 & \rho_1 & \rho_2 & \rho_3 & \rho_1^2 & \rho_1\rho_2 & \rho_1\rho_3 & \rho_2^2 & \rho_2\rho_3 & \rho_3^2 \\ \hline x_1^2 & -2x_1R_1 & & & R_1^2 & & & & & \\ x_1x_2 & -x_2R_1 & -x_1R_2 & & & R_1R_2 & & & & \\ x_1x_3 & -x_3R_1 & & -x_1R_3 & & & R_1R_3 & & & \\ x_2^2 & & -2x_2R_2 & & & & & R_2^2 & & \\ x_2x_3 & & -x_3R_2 & -x_2R_3 & & & & & R_1R_3 & \\ x_3^2 & & & -2x_3R_3 & & & & & & R_2^2 \end{array}\right) \begin{array}{l} \\ \leftarrow (x_1 - r_1)^2 \\ \leftarrow (x_1 - r_1)(u_2 - r_2) \\ \leftarrow (x_1 - r_1)(u_3 - r_3) \\ \leftarrow (x_2 - r_2)^2 \\ \leftarrow (x_2 - r_2)(u_3 - r_3) \\ \leftarrow (x_3 - r_3)^2 \end{array}$$

As before, we consider the lattice spanned by the rows of this matrix, and look for vectors with small l_1 norm in this lattice. Any vector $\langle v_0, v_1, \ldots, v_9 \rangle$ in this lattice corresponds to a multivariate polynomial of total degree two in the ρ_i's, whose value at the point $\left\langle \rho_1 = \frac{r_1}{R_1}, \rho_2 = \frac{r_2}{R_2}, \rho_3 = \frac{r_3}{R_3} \right\rangle$ is equal to zero modulo p^2. If in addition the l_1 norm of this vector is smaller than p^2, then it must be the case that the polynomial evaluates to zero over the reals. If we can find t such small vectors (corresponding to independent polynomials) then we can use resultants to eliminate all but one of the variables, and then find the roots of the last polynomial, which would give one of the r_i's (and therefore also p and all the other r_i's).

To get an estimate on the length of the vectors that we can expect to find, we again need to estimate the determinant of the lattice. We note that in each of the columns, all the non-zero entries are likely roughly the same size (up to polynomial factor in t). Hence if we use elementary row operations to put this matrix in row-echelon form, then the coefficients of these elementary operations would probably all be polynomially small. The resulting matrix (whose determinant is equal to that of M) would have this form:

$$\begin{pmatrix} \tilde{O}(X^2) & 0 & 0 & 0 & 0 & 0 & \tilde{O}(R^2) & \tilde{O}(R^2) & \tilde{O}(R^2) & \tilde{O}(R^2) \\ 0 & \tilde{O}(XR) & 0 & 0 & 0 & 0 & \tilde{O}(R^2) & \tilde{O}(R^2) & \tilde{O}(R^2) & \tilde{O}(R^2) \\ 0 & 0 & \tilde{O}(XR) & 0 & 0 & 0 & \tilde{O}(R^2) & \tilde{O}(R^2) & \tilde{O}(R^2) & \tilde{O}(R^2) \\ 0 & 0 & 0 & \tilde{O}(XR) & 0 & 0 & \tilde{O}(R^2) & \tilde{O}(R^2) & \tilde{O}(R^2) & \tilde{O}(R^2) \\ 0 & 0 & 0 & 0 & \tilde{O}(R^2) & 0 & \tilde{O}(R^2) & \tilde{O}(R^2) & \tilde{O}(R^2) & \tilde{O}(R^2) \\ 0 & 0 & 0 & 0 & 0 & \tilde{O}(R^2) & \tilde{O}(R^2) & \tilde{O}(R^2) & \tilde{O}(R^2) & \tilde{O}(R^2) \end{pmatrix}$$

The determinant can now be estimated as roughly equal to the product of the row-lengths, and estimating the row-length using the size of the pivot we get

$$\det(M) \;\approx\; X^2 \cdot (XR)^t \cdot (R^2)^{\binom{t}{2}-1} = X^{2+t} R^{t^2-1}.$$

Since the lattice $L(M)$ has dimension $T = \binom{t+1}{2}$, we expect to be able to find vectors in it of size about

$$2^{\varepsilon T} \cdot \sqrt[T]{\det(M)} \approx 2^{\varepsilon t(t+1)/2} \cdot (R(X^{(t+2)/(t+1)}/R)^{1/t})^2,$$

which is $\gg p^2$ for $t \leq (\gamma - \rho)/(\eta - \rho)$. The rest of the analysis is similar to that for the previous attacks. (Note that here we actually do worse with products of pairs of relations than we did with the individual relations, in that the size of the vector that we expect grew from $2^{O(t^{3/2})}$ to $2^{O(t^2)}$.)

The general case. The heuristic analysis from above can be extended to the product of any number of relations. In the general case, we use all the products of exactly d relations (out of the t available ones, with repetitions). Hence we get $\binom{t+d-1}{d}$ relations, defined over $\binom{t+d}{d+1}$ terms, all holding modulo p^d. Putting all these relations in a matrix, we have an $\binom{t+d-1}{d} \times \binom{t+d}{d+1}$ matrix with each row corresponding to a relation and each column corresponding to a term. Since all the x_i's are roughly the same size X, and we have the same bound R on all the r_i's, then in a column corresponding to a term of degree $i \leq d$, every nonzero entry is of size roughly $X^{d-i}R^i$. Hence we can do elementary row operations with small coefficients to put this matrix in row-echelon form, and the size of the pivot in a column corresponding to a degree-i term will still be $\tilde{O}(X^{d-i}R^i)$.

Using these pivots as estimates for the row size, we have one row of size $\tilde{O}(X^d)$, t rows of size $\tilde{O}(X^{d-1}R)$, etc. In general for each $i < d$ we have $\binom{t+i-1}{i}$ rows of size $\tilde{O}(X^{d-i}R^i)$, and the remaining rows are of size $\tilde{O}(R^d)$ (there are $\binom{t+d-1}{d} - \sum_{i=0}^{d-1} \binom{t+i-1}{i}$ of those).

C Circuit Privacy

As described so far, our scheme may leak some information about the circuit to the holder of the secret key. For example, in the somewhat homomorphic scheme from Section 3 even just the bit-length of the ciphertext reveals the number of multiplication operations in the circuit. So if we use the "augmented decryption circuits", it reveals whether the final operation after encryption was addition or multiplication.[8]

If we are using the ciphertext-size-reduction optimization from Section 3.3 then it is easy to re-randomize the ciphertext so as to remove all traces of the circuit, by adding a "high noise" encryption of zero. For the Somewhat Homomorphic scheme from Section 3, this means using a somewhat larger value of η (call it η'), so that evaluating permitted circuits result in noise of bit length at most $\eta' - \omega(log\lambda)$. After computing the final ciphertext c (with respect to public key $\text{pk} = \langle x_0, x_1, \dots, x_\tau \rangle$) we choose a random subset $S \subseteq_R \{1, \dots, \tau\}$ and a random noise integer, say $r \leftarrow [-2^{\eta'-5},\ 2^{\eta'-5}]$, and output $c' \leftarrow [c + \sum_{i \in S} x_i]_{x_0} + 2r$. Clearly, the added term $2r$ hides the noise component of the original ciphertext c, making $r_p(c')$ close to uniform. At the same time, adding the subset $\sum_{i \in S} x_i$ randomizes the quotient, making $q_p(c')$ close to uniform modulo $q_p(x_0)$ (again by the leftover hash lemma).

It is clear that this technique only works if the ciphertext size before re-randomization is not much larger than x_0 (or else the mod x_0 operation will introduce too much noise). Hence the requirement of using the ciphertext-size-reduction optimization. If we do not use the ciphertext-size-reduction optimization, then we can add to the public key "big encryptions of 0" for this purpose. (Namely, integers $x' = q'p + r'$ with r' in the usual range $[-2^\rho, 2^\rho]$ but q' of size as large as one gets as the result of $\mathsf{Evaluate}$.)

Alternatively, it is always possible to convert a compact homomorphic encryption scheme to one that offers also circuit privacy, by using Yao's garbled circuit to homomorphically perform the decryption operation of the compact scheme.

We note that all of these solutions apply only to the "honest-but-curious" case, where the public key is generated as it should by the KeyGen procedure. We did not look into the problem of hiding the circuit in the presence of a maliciously-generated public key.

D How to Compress Our Ciphertexts

Ciphertexts in our somewhat homomorphic scheme are very long – $\tilde{\theta}(\lambda^5)$ bits under our suggested parameters. Here, we show how to "compress" or post-process the ciphertexts from the somewhat homomorphic scheme down to (asymptotically) the size of an RSA modulus, reducing the communication complexity of our scheme dramatically.

The best time to use the compression algorithm is on the final output ciphertexts, after all applications of the $\mathsf{Evaluate}$ algorithm have been completed. The reason is that the compressed ciphertexts cannot be decrypted by a shallow circuit, and consequently the modified somewhat homomorphic scheme is not bootstrappable.

Abstractly, the modified scheme works as follows. We supplement the public key with the description of a cyclic group $G = \langle g \rangle$ whose order is a multiple of p. The compressed ciphertext

[8]This particular problem can be solved by canonicalizing the circuits, ensuring that they all have the same number of additions and multiplication (e.g., by adding multiplications by one and additions of zero as needed). Still, this may not be enough to ensure that no information about the circuit is leaking.

is simply $c^* \leftarrow g^c$. Note that $\mathrm{DL}_g(c^*) = c \bmod p$. To decrypt the compressed ciphertext, one sets $m \leftarrow (\mathrm{DL}_g(c^*) \bmod p) \bmod 2$. Correctness follows immediately from the correctness of the original scheme. We prove that the modified scheme is semantically secure, if the approximate-gcd problem is hard and the following *decision subgroup problem* (informally stated here) is hard: for distinct fixed integers $p^{(0)}, p^{(1)}$, given the description of a group G whose order is divisible by either $p^{(0)}$ or $p^{(1)}$, decide which is the case.

To make the abstract approach concrete, first we require p to be *smooth* – specifically, a product of prime numbers that are all polynomial in λ. During key generation, one samples a modulus N such that N "phi-hides" p – i.e., such that $p|\phi(N)$ – along with a generator g of a cyclic subgroup of $\mathbb{Z}_N^*$ whose order is a multiple of p. The encrypter or evaluator computes c as in the somewhat homomorphic scheme, and sets $c^* \leftarrow g^c \bmod N$. The decrypter uses its knowledge of the factorization of N to compute the discrete log $\mathrm{DL}_g(c^*) \bmod p$. The smoothness of p allows the decrypter to compute this discrete log efficiently. Of course, security relies in part on the approximate-gcd problem still being hard for this particular (smooth) distribution of p.

Our compressed ciphertext is $\log N$ bits. For the decision subgroup problem to be hard, we need $(\log N)/(\log p) > 4$; otherwise, Coppersmith's method can be used to factor N efficiently [5]. However, if the ratio is a bit larger – e.g., if $(\log N)/(\log p) > 10$ – the best known attack on the decision subgroup problem is to factor N using the number field sieve (NFS). This attack takes exponential (i.e., 2^λ) time when $\log N$ is $\theta(\lambda^3/\log^2 \lambda)$, which is $\omega(\log p)$ for our suggested parameters. In particular, asymptotically, the integer N – and hence a compressed ciphertext – has the same size as an RSA modulus.

We now describe the modified scheme more formally. Let $\mathcal{G}$ be a set of cyclic groups. Let $\mathsf{GroupGen}(\mathcal{G}, p, \ell)$ be an algorithm that takes as input $\mathcal{G}$ and $p < 2^\ell$ for $\ell = \mathrm{poly}(\lambda)$, and samples a group G from $\mathcal{G}$ with generator g whose order n_g is at least 2^ℓ and divisible by p; it outputs (G, g, n_g). Later, we specify how to instantiate $\mathsf{GroupGen}$ over composite moduli, and the parameter ℓ will concretely represent the bit-length of the modulus N, in which case we may have $\ell = \Omega(\lambda^3/\log^2 \lambda)$.

$\mathsf{KeyGen}_{\mathcal{E}^*}(\lambda)$: Run $(\mathrm{sk}, \mathrm{pk}) \leftarrow \mathsf{KeyGen}_{\mathcal{E}}(\lambda)$, but with the requirement that p is $\mathrm{poly}(\lambda)$-smooth. Run $(G, g, n_g) \leftarrow \mathsf{GroupGen}(p, \ell)$. Output the secret key $\mathrm{sk}^* \leftarrow (\mathrm{sk}, n_g)$ and the public key $\mathrm{pk}^* \leftarrow (\mathrm{pk}, G, g)$.

$\mathsf{Compress}_{\mathcal{E}^*}(\mathrm{pk}^*, c)$: Takes as input pk and a ciphertext c output by $\mathsf{Encrypt}_{\mathcal{E}}$ or $\mathsf{Evaluate}_{\mathcal{E}}$. Output the ciphertext $c^* \leftarrow g^c \in G$.

$\mathsf{DecryptComp}_{\mathcal{E}^*}(\mathrm{sk}^*, c^*)$: For each prime divisor p_i of p, output set $x_i \leftarrow \mathrm{DL}_{g^{n_g/p_i}}(c^{*n_g/p_i}) \bmod p_i$. Use CRT to compute $x = \mathrm{DL}_g(c^*) \bmod p$ from the x_i's. Output $x \bmod 2$.

Remark D.1. Regarding $\mathsf{KeyGen}_{\mathcal{E}^*}$, the t-th prime number p_t satisfies $p_t = (1+o(1))t \ln t$, and the product of the first t prime numbers – i.e., the "primorial" $p_t\#$ – is $\exp((1+o(1))t \ln t)$. Therefore, asymptotically, to sample a smooth p of $\eta = \tilde{\theta}(\lambda^2)$ bits, we can set p to be a random subset product of the prime numbers less than $2\eta = \mathrm{poly}(\lambda)$.

Remark D.2. Regarding $\mathsf{DecryptComp}_{\mathcal{E}^*}$, note that the group generated by g^{n_g/p_i} has small order – namely, $p_i = \mathrm{poly}(\lambda)$. Computing $\mathrm{DL}_{g^{n_g/p_i}}(c^{*n_g/p_i}) \bmod p_i$ takes only $\sqrt{p_i}$ arithmetic operations using the baby-step-giant-step method.

We formally define the decision subgroup problem as in [8]. This problem is closely related to the phi-hiding problem, described in [3].

Definition D.3 (Decision Subgroup Problem (from [8]))**.** Let η, ℓ be parameters. Let $\pi_0, \pi_1 \in [2^{\eta-1}, 2^{\eta}-1]$ be distinct integers. Let $\mathcal{G}$ be the set of cyclic groups, and $\mathsf{GroupGen}^{\dagger}(\mathcal{G}, \pi, \ell)$ be an algorithm that samples a group G from $\mathcal{G}$ with generator g whose order is in $[2^{\ell}, 2^{\ell+1}-1]$ and divisible by π, and outputs (G, g). We say that algorithm A has advantage ϵ against the $(\mathcal{G}, \eta, \ell, \pi_0, \pi_1, \mathsf{GroupGen}^{\dagger})$-Decision Subgroup Problem if

$$\left| \Pr\left[b \xleftarrow{R} \{0,1\}, G_b \xleftarrow{D} \mathsf{GroupGen}^{\dagger}(\mathcal{G}, \pi_b, \ell) : A(\mathcal{G}, G_b, \eta, \ell, \pi_0, \pi_1, \mathsf{GroupGen}^{\dagger}) = b \right] - \frac{1}{2} \right| \geq \epsilon.$$

Definition D.4 (Decision Subgroup Assumption)**.** For appropriate η, ℓ, and $\mathsf{GroupGen}^{\dagger}$, the decision subgroup problem is hard for any π_0, π_1.

In the following theorem, we let $\mathcal{E}$ be the statistically circuit-private version of the somewhat homomorphic scheme, and let $\mathcal{E}^*$ refer to the modified scheme.

Theorem D.5. *Suppose that there is an algorithm $\mathcal{A}$ that breaks the semantic security $\mathcal{E}^*$ with advantage ϵ. Let $\mathsf{GroupGen}^{\dagger}$ be identical to the algorithm $\mathsf{GroupGen}$ used in $\mathcal{E}^*$, except that it does not output the order n_g. Then, there are algorithms $\mathcal{B}_0$ and $\mathcal{B}_1$, each running in about the same time as $\mathcal{A}$, such that either $\mathcal{B}_0$ has advantage $\epsilon/3$ in solving the decision subgroup problem or $\mathcal{B}_1$ has advantage $\epsilon/3$ against the semantic security of $\mathcal{E}$.*

Proof. Let Game 0 be the real-world semantic security game against the modified encryption scheme, where the challenge is a modified ciphertext. Note that the $\mathcal{E}$ ciphertext from which the challenge modified ciphertext is derived could be an output of $\mathsf{Encrypt}_{\mathcal{E}}$ or $\mathsf{Evaluate}_{\mathcal{E}}$, but by the statistical circuit privacy of $\mathcal{E}$, the output distributions of $\mathsf{Encrypt}_{\mathcal{E}}$ and $\mathsf{Evaluate}_{\mathcal{E}}$ are statistically identical, conditional on the underlying plaintexts being identical.

Game 1 is like Game 0, except the challenger generates pk^* differently. It runs $(\mathrm{sk}, \mathrm{pk}) \leftarrow \mathsf{KeyGen}_{\mathcal{E}}$ as before, where sk is the prime integer $p = p^{(0)}$. But instead of inputting p into the $\mathsf{GroupGen}$ algorithm, it generates a random η-bit odd number $p^{(1)}$ and runs $(G, g, n_g) \leftarrow \mathsf{GroupGen}(p^{(1)}, \ell)$, and sets $\mathrm{pk}^* \leftarrow (\mathrm{pk}, G, n_g)$. By assumption, ϵ is $\mathcal{A}$'s advantage in Game 0. Let ϵ' be $\mathcal{A}$'s advantage in Game 1.

$\mathcal{B}_0$ runs as follows. It runs $(\mathrm{sk}, \mathrm{pk}) \leftarrow \mathsf{KeyGen}_{\mathcal{E}}(\lambda)$, sets $p^{(0)} \leftarrow \mathrm{sk}$, and generates random η-bit odd number $p^{(1)}$. It then asks for a decision subgroup problem instance with respect to $p^{(0)}$ or $p^{(1)}$. The challenger sets bit $b \xleftarrow{\mathrm{R}} \{0,1\}$ and sends a decision subgroup instance (G, g) to $\mathcal{B}_0$ with respect to $p^{(b)}$. $\mathcal{B}_0$ sets $\mathrm{pk}^* \leftarrow (\mathrm{pk}, G, g)$ and sends pk^* to $\mathcal{A}$. When $\mathcal{A}$ asks for a challenge ciphertext on one of (m_0, m_1), $\mathcal{B}_0$ sets $\beta \xleftarrow{\mathrm{R}} \{0,1\}$, sets $c \leftarrow \mathsf{Encrypt}_{\mathcal{E}}(\mathrm{pk}, m_\beta)$, and sends $c^* \leftarrow g^c$. Eventually, $\mathcal{A}$ sends a bit β'. $\mathcal{B}_0$ sends $b' \leftarrow \beta \oplus \beta'$ to the challenger. Note that the public key pk (and the other aspects of the simulation) is distributed exactly as in Game b. In particular, by the statistical circuit privacy of $\mathcal{E}$, the real-world distribution of the challenge is the same. Therefore, we compute that $\mathcal{B}_0$'s advantage is at least $|\epsilon - \epsilon'|/2$.

$\mathcal{B}_1$ attacks the semantic security of the original scheme $\mathcal{E}$ as follows. It obtains an $\mathcal{E}$ public key pk from the challenger. It generates random η-bit odd number $p^{(1)}$, runs $(G, g, n_g) \leftarrow \mathsf{GroupGen}(p^{(1)}, \ell)$, and sends $\mathrm{pk}^* \leftarrow (\mathrm{pk}, G, g)$ to $\mathcal{A}$. When $\mathcal{A}$ asks for a challenge ciphertext on one of (m_0, m_1), $\mathcal{B}_1$ asks the challenger for a challenge ciphertext on one of (m_0, m_1). The challenger sends back a challenge ciphertext c. $\mathcal{B}_1$ sends $c^* \leftarrow g^c$ to $\mathcal{A}$. $\mathcal{A}$ sends a bit b', which $\mathcal{B}_1$ forwards to the challenger. We see that the distribution is the same as in Game 1. Also, $\mathcal{B}_1$'s bit is correct if $\mathcal{A}$'s bit is correct. So $\mathcal{B}_1$ has advantage ϵ'.

The theorem follows. □

We can implement $\mathsf{GroupGen}$ as in prior work [3, 8]. For example, Gentry and Ramzan [8] generate a $N = Q_0Q_1$, where $Q_0 = 2q_0p + 1$ is a random "semi-safe" prime with q_0 prime, and $Q_1 = 2dq_1 + 1$ for prime q_1 and d chosen uniformly from a large interval. This technique ensures that $N \bmod p$ is statistically uniform. They show that, for this instantiation of $\mathsf{GroupGen}$, the best *generic* attacks on the decision subgroup assumption take exponential time. We refer to [3, 8] for the details.