Faster Bootstrapping with Polynomial Error

Jacob Alperin-Sheriff and Chris Peikert*

School of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA

Abstract. *Bootstrapping* is a technique, originally due to Gentry (STOC 2009), for "refreshing" ciphertexts of a somewhat homomorphic encryption scheme so that they can support further homomorphic operations. To date, bootstrapping remains the only known way of obtaining fully homomorphic encryption for arbitrary unbounded computations.

Over the past few years, several works have dramatically improved the efficiency of bootstrapping and the hardness assumptions needed to implement it. Recently, Brakerski and Vaikuntanathan (ITCS 2014) reached the major milestone of a bootstrapping algorithm based on Learning With Errors for *polynomial* approximation factors. Their method uses the Gentry-Sahai-Waters (GSW) cryptosystem (CRYPTO 2013) in conjunction with Barrington's "circuit sequentialization" theorem (STOC 1986). This approach, however, results in *very large* polynomial runtimes and approximation factors. (The approximation factors can be improved, but at even greater costs in runtime and space.)

In this work we give a new bootstrapping algorithm whose runtime and associated approximation factor are both *small* polynomials. Unlike most previous methods, ours implements an elementary and efficient *arithmetic* procedure, thereby avoiding the inefficiencies inherent to the use of boolean circuits and Barrington's Theorem. For 2^λ security under conventional lattice assumptions, our method requires only a *quasi-linear* $\tilde{O}(\lambda)$ number of homomorphic operations on GSW ciphertexts, which is optimal (up to polylogarithmic factors) for schemes that encrypt just one bit per ciphertext. As a contribution of independent interest, we also give a technically simpler variant of the GSW system and a tighter error analysis for its homomorphic operations.

1 Introduction

Gentry's *bootstrapping* paradigm [11, 10] allows for converting a "somewhat homomorphic" encryption scheme (which supports only a bounded number of homomorphic operations) into a fully homomorphic encryption one (which has no

* This material is based upon work supported by the National Science Foundation under CAREER Award CCF-1054495, by the Alfred P. Sloan Foundation, and by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under Contract No. FA8750-11-C-0098. The views expressed are those of the authors and do not necessarily reflect the official policy or position of the National Science Foundation, the Sloan Foundation, DARPA or the U.S. Government.

J.A. Garay and R. Gennaro (Eds.): CRYPTO 2014, Part I, LNCS 8616, pp. 297–314, 2014.

such bound). The bounded nature of all known somewhat-homomorphic schemes is an artifact of "error" terms in their ciphertexts, which are necessary for security. The error grows as a result of performing homomorphic operations, and if it grows too large, the ciphertext will no longer decrypt correctly.

Bootstrapping "refreshes" a ciphertext—i.e., reduces its error—so that it can support more homomorphic operations. This is accomplished by *homomorphically evaluating* the decryption function on the ciphertext. The result is a ciphertext that still encrypts the original encrypted message, and moreover, as long as the error incurred in the homomorphic evaluation is smaller than the error in the original ciphertext, the ciphertext is "refreshed." To date, the bootstrapping paradigm is the only known way of obtaining an *unbounded* FHE scheme, i.e., one that can homomorphically evaluate any efficient function using keys and ciphertexts of a fixed size. (By contrast, *leveled* FHE schemes can evaluate functions of any *a priori* bounded depth, and can be constructed without resorting to bootstrapping [4].)

Bootstrapping has received intensive study, with progress often going hand-in-hand with innovations in the design of homomorphic encryption schemes, e.g., [12, 6, 4, 14, 13, 1, 15, 7]. Of particular interest is a recent major milestone due to Brakerski and Vaikuntanathan (BV) [7], who gave a bootstrapping method that incurs only *polynomial* error in the security parameter λ. This allows security to be based on the learning with errors (LWE) problem [20] with inverse-polynomial error rates, and hence on worst-case lattice problems with polynomial approximation factors (via the reductions of [20, 19, 5]). The BV method is centered around two main components:

1. the recent homomorphic cryptosystem of Gentry, Sahai, and Waters (GSW) [15], specifically, the "*quasi-additive*" nature of its error growth under homomorphic multiplication; and
2. the "circuit sequentialization" property of Barrington's Theorem [3], which converts any depth-d circuit (of NAND gates) into a length-4^d "branching program," which is essentially a fixed sequence of conditional multiplications.

Since decryption in homomorphic cryptosystems can be implemented in circuit depth $O(\log \lambda)$, Barrington's Theorem yields an equivalent branching program of length $4^d = \mathrm{poly}(\lambda)$. Moreover, the quasi-additive error growth of GSW multiplication means that homomorphic evaluation of the branching program incurs only $\mathrm{poly}(\lambda)$ error, as demonstrated in [7].

The polynomial error growth of the BV bootstrapping algorithm is a terrific feature, but the method also has two significant drawbacks: it comes at a high price in *efficiency*, and the error growth is a *large* polynomial. Both issues arise from the fact that in this context, Barrington's Theorem yields a branching program of large polynomial length. Existing analyses (e.g., [6, Lemma 4.5]) of decryption circuits (for cryptosystems with 2^λ security) yield depths of $c \log \lambda$ for some unspecified but moderately large constant $c \geq 3$, which translates to a branching program of length at least $\lambda^{2c} \geq \lambda^6$. (Even if the depth were to be improved, there is a fundamental barrier of $c \geq 1$, which yields length $\Omega(\lambda^2)$.) The branching program length is of course a lower bound on the number of

homomorphic operations required to bootstrap, and it also largely determines the associated error growth and final lattice approximation factors.

Separately, Brakerski and Vaikuntanathan also show how to obtain better lattice approximation factors through a kind of "dimension leveraging" technique, but this comes at an even higher price in efficiency: if the original error growth was λ^c for some constant c, then the technique involves running the bootstrapping procedure with GSW ciphertexts of dimension $n \approx \lambda^{c/\epsilon}$, where the choice of $\epsilon \in (0, 1)$ yields a final approximation factor of $\tilde{O}(n^{3/2+\epsilon})$. The high cost of dimension leveraging underscores the importance of obtaining smaller error growth and approximation factors via other means.

1.1 Our Results

Our main result is a new bootstrapping method having substantially smaller runtime *and* (polynomial) error growth than the recent one from [7]. The improvements come as a result of treating decryption as an *arithmetic* function, in contrast to most earlier works which treated decryption as a boolean circuit. This avoids the circuitous and inefficient path of constructing a shallow circuit and then transforming it via Barrington's Theorem into a branching program of (large) polynomial length. Instead, we show how to *directly* evaluate the decryption function in an elementary and efficient arithmetic form, using just basic facts about cyclic groups. See the next subsection for a detailed overview.

Our method requires only a *quasi-linear* $\tilde{O}(\lambda)$ number of homomorphic operations on GSW ciphertexts, to bootstrap essentially any LWE-based encryption scheme with 2^λ security under conventional assumptions. This performance is *quasi-optimal* (i.e., ignoring polylogarithmic factors) for a system with bitwise encryption (like GSW), because the decryption function must depend on at least λ secret key bits. When instantiated with a GSW scheme based on ring-LWE [17], in which the cost of each homomorphic operation is only $\tilde{O}(\lambda)$ bit operations, the total runtime of our algorithm is a respectable $\tilde{O}(\lambda^2)$ bit operations.[1]

Regarding error growth, the security of our basic scheme can be based on LWE with error rates as large as $1/\tilde{O}(\lambda \cdot n)$, where $n = \tilde{\Omega}(\lambda)$ is the LWE dimension used in the GSW scheme. Taking $n = \tilde{O}(\lambda)$ to be asymptotically minimal, this translates to lattice approximation factors of $\tilde{O}(n^3)$, which is quite close to the $\tilde{O}(n^{3/2})$ factors that plain public-key encryption can be based upon (and quite a bit smaller than for many other applications of LWE!). We emphasize that these small factors are obtained *directly* from our construction with *small* LWE dimensions. To further improve the assumptions at a (high) cost in efficiency, we can let $n = \lambda^{1/\epsilon}$ to directly yield $\tilde{O}(n^{2+\epsilon})$ factors for any $\epsilon \in (0, 1)$, or we can use the successive dimension/modulus-reduction technique from [7] to obtain $\tilde{O}(n^{3/2+\epsilon})$ factors.

Simpler GSW variant. As a contribution of independent interest, we also give a variant of the GSW cryptosystem that we believe is technically simpler, along

[1] Homomorphic operations in standard-LWE-based GSW are quite a bit more expensive, due to matrix multiplications of dimensions exceeding λ.

with a tighter analysis of error terms under its homomorphic operations (see Section 3). The entire scheme, security proof, and error analysis fit into just a few lines of standard linear algebra notation, and our variant enjoys additional useful properties like full "re-randomization" of error terms as a natural side effect. The error analysis is also very clean and tight, due to the use of *subgaussian* random variables instead of coarser measures like the ℓ_2 or ℓ_∞ norms. One nice consequence of this approach is that the error in a homomorphic product of d ciphertexts grows with $\sqrt{d}$, rather than linearly as in prior analyses. This is important for establishing the small error growth of our bootstrapping method.

1.2 Technical Overview

Here we give an overview of the main ideas behind our new bootstrapping method. We start by recalling in more detail the main ideas behind the work of Brakerski and Vaikuntanathan [7], which uses the Gentry-Sahai-Waters (GSW) [15] homomorphic encryption scheme to obtain FHE from LWE with inverse-polynomial error rates, and hence from lattice problems with polynomial approximation factors.

The starting point is a simple observation about the GSW encryption scheme: for encryptions $\mathbf{C}_1, \mathbf{C}_2$ of messages $\mu_1, \mu_2 \in \mathbb{Z}$, the error in the homomorphic product $\mathbf{C}_1 \boxdot \mathbf{C}_2$ of $\mu_1 \cdot \mu_2$ is *"quasi-additive"* and *asymmetric* in the ciphertexts' respective errors e_1, e_2, namely, it is $e_1 \cdot \mathrm{poly}(n) + \mu_1 \cdot e_2$, where n is the dimension of the ciphertexts. (The error in the homomorphic sum $\mathbf{C}_1 \boxplus \mathbf{C}_2$ is simply the sum $e_1 + e_2$ of the individual errors.) This property has a number of interesting consequences. For example, Brakerski and Vaikuntanathan use it to show that the homomorphic product of many freshly encrypted 0-1 messages, if evaluated *sequentially* in a right-associative manner, has error that grows at most linearly in the number of ciphertexts. More generally, the homomorphic product of many encrypted *permutation* matrices—i.e., 0-1 matrices in which each row and column has exactly one nonzero entry—has similarly small error growth.

The next main idea from [7] is to use Barrington's Theorem to express the boolean decryption circuit of depth $d = O(\log \lambda)$ as a branching program of length $4^d = \mathrm{poly}(\lambda)$ over the symmetric group S_5, or equivalently, the multiplicative group of 5-by-5 permutations matrices. Their bootstrapping algorithm homomorphically (and sequentially) multiplies appropriate encrypted permutation matrices to evaluate this branching program on a given input ciphertext, thereby homomorphically decrypting it. Since evaluation is just a homomorphic product of $\mathrm{poly}(\lambda)$ permutation matrices, the error in the final output ciphertext is only polynomial, and the LWE parameters can be set to yield security assuming the hardness of lattice problems for polynomial approximation factors.

Our Approach. Our bootstrapping method retains the use of symmetric groups and permutation matrices, but it works without the "magic" of Barrington's Theorem, by treating decryption more directly and efficiently as an *arithmetic*

function, not a boolean circuit. In more detail, the decryption function for essentially every LWE-based cryptosystem can without loss of generality (via standard bit-decomposition techniques) be written as a "rounded inner product" between the secret key $\mathbf{s} \in \mathbb{Z}_q^d$ and a *binary* ciphertext $\mathbf{c} \in \{0,1\}^d$, as

$$\mathsf{Dec}(\mathbf{s}, \mathbf{c}) = \lfloor \langle \mathbf{s}, \mathbf{c} \rangle \rceil_2 \in \{0,1\}.$$

Here the modular rounding function $\lfloor \cdot \rceil_2 \colon \mathbb{Z}_q \to \{0,1\}$ indicates whether its argument is "far from" or "close to" 0 (modulo q), and the dimension d and modulus q can both be made as small as quasi-linear $\tilde{O}(\lambda)$ in the security parameter via dimension/modulus reduction [6], while still providing provable 2^λ security under conventional lattice assumptions. Note that the inner product itself is just a subset-sum of the $\mathbb{Z}_q$-entries of $\mathbf{s}$ indicated by $\mathbf{c}$, and uses only the additive group structure of $\mathbb{Z}_q$.

Embedding $\mathbb{Z}_q$ into S_q. As a warm up, we first observe that the additive group $\mathbb{Z}_q$ embeds (i.e., has an injective homomorphism) into the symmetric group S_q, the multiplicative group of q-by-q permutation matrices. (This is just a special case of Cayley's Theorem, which says that any finite group G embeds into $S_{|G|}$.) The embedding is very simple: $x \in \mathbb{Z}_q$ maps to the permutation that cyclically rotates by x positions. Moreover, any such permutation can be represented by an indicator vector in $\{0,1\}^q$ with its 1 in the position specified by x, and its permutation matrix is obtained from the cyclic rotations of this vector. In this representation, a sum $x + y$ can be computed in $O(q^2)$ bit operations by expanding x's indicator vector into its associated permutation matrix, and then multiplying by y's indicator vector. This representation also makes the *rounding function* $\lfloor \cdot \rceil_2 \colon \mathbb{Z}_q \to \{0,1\}$ trivial to evaluate: one just sums the entries of the indicator vector corresponding to those values in $\mathbb{Z}_q$ that round to 1.

These ideas already yield a new and simple bootstrapping algorithm that appears to have better runtime and error growth than can be obtained using Barrington's Theorem. The bootstrapping key is an encryption of each coordinate of the secret key $\mathbf{s} \in \mathbb{Z}_q^d$, represented as a dimension-q indicator vector, for a total of $d \cdot q = \tilde{O}(\lambda^2)$ GSW ciphertexts. To bootstrap a ciphertext $\mathbf{c} \in \{0,1\}^d$, the inner product $\langle \mathbf{s}, \mathbf{c} \rangle \in \mathbb{Z}_q$ is computed homomorphically as a subset-sum using the addition method described above, in $O(d \cdot q^2) = \tilde{O}(\lambda^3)$ homomorphic operations. The rounding function is then applied homomorphically, using just $O(q) = \tilde{O}(\lambda)$ additions.

Embedding $\mathbb{Z}_q$ into smaller symmetric groups. While the above method yields some improvements over prior work, it is still far from optimal. Our second main idea is an efficient way of embedding $\mathbb{Z}_q$ into a *much smaller* symmetric group S_r for some $r = \tilde{O}(1)$, such that the rounding function can still be efficiently evaluated (homomorphically). We do so by letting the modulus $q = \prod_i r_i$ be the product of many small prime powers r_i of distinct primes. (We can use such a q by modulus switching, as long as it remains sufficiently large to preserve correctness of decryption.) Using known bounds on the distribution of primes,

it suffices to let the r_i be maximal prime powers bounded by $O(\log \lambda)$, of which there are at most $O(\log \lambda / \log\log \lambda)$.

By the Chinese Remainder Theorem, the additive group $\mathbb{Z}_q$ is isomorphic (via the natural homomorphism) to the product group $\prod_i \mathbb{Z}_{r_i}$, which then embeds into $\prod_i S_{r_i}$ as discussed above. Therefore, we can represent any $x \in \mathbb{Z}_q$ as a tuple of $O(\log \lambda)$ indicator vectors of length $r_i = O(\log \lambda)$ representing $x \pmod{r_i}$, and can perform addition by operating on the indicator vectors as described above. In this representation, the rounding function is no longer just a sum, but it can still be expressed relatively simply as

$$\lfloor x \rceil_2 = \sum_{v \in \mathbb{Z}_q \text{ s.t. } \lfloor v \rceil_2 = 1} [x = v],$$

where each equality test $[x = v]$ returns 0 for false and 1 for true.[2] In turn, each equality test $[x = v]$ is equivalent to the product of equality tests $[x = v \pmod{r_i}]$, each of which can be implemented trivially in our representation by selecting the appropriate entry of the indicator vector for $x \pmod{r_i}$. All of these operations have natural homomorphic counterparts in our representation, so we get a corresponding bootstrapping algorithm.

As a brief analysis, each coordinate of the secret key $\mathbf{s} \in \mathbb{Z}_q^d$ is encrypted as $\sum_i r_i = \tilde{O}(1)$ GSW ciphertexts, for a total of $\tilde{O}(d) = \tilde{O}(\lambda)$ ciphertexts in the bootstrapping key. Similarly, each addition or equality test over $\mathbb{Z}_q$ takes $\tilde{O}(1)$ homomorphic operations, for a total of $\tilde{O}(d+q) = \tilde{O}(\lambda)$. Both of these measures are quasi-optimal when relying on a scheme that encrypts one bit per ciphertext (like GSW). By contrast, bootstrapping using Barrington's Theorem requires at least $4^{c \log \lambda} = \lambda^{2c}$ homomorphic operations to evaluate the branching program, where $c \log \lambda$ is the depth of the decryption circuit *using NAND gates* (of fan-in 2). To our knowledge, upper bounds on the constant c have not been optimized or even calculated explicitly, but existing analyses like [6, Lemma 4.5] yield $c \gg 3$, and the necessary dependence on λ inputs bits for 2^λ security yields a fundamental barrier of $c \geq 1$.

Related Work on Branching Programs. Several works have extended and improved Barrington's Theorem for the simulation of general circuits and formulas via branching programs, e.g., [8, 9]. Of particular interest here is the thesis of Sinha [22], which gave quasi-linear-size, log-width branching programs for threshold functions (i.e., those which output 1 if at least some k of the n inputs are 1) and "mod" functions (i.e., those which output 1 if the number of 1s in the input is zero modulo some d). Similarly to our techniques, Sinha's construction uses the Chinese Remainder Theorem over many small primes in an essential way.

Because decryption in LWE-based cryptosystems involves modular addition, and can be implemented in constant depth (and polynomial size) by threshold

[2] Note that we are not using any special property of the rounding function here; any boolean function $f \colon \mathbb{Z}_q \to \{0,1\}$ can be expressed similarly by summing over $f^{-1}(1)$.

gates, it might be possible to bootstrap in a quasi-linear number of homomorphic operations by using Sinha's results in place of Barrington's Theorem. However, we have not seen a way to make this work concretely.

Organization. The rest of the paper is organized as follows. In Section 2 we recall some mathematical preliminaries on subgaussian random variables and symmetric groups. In Section 3 we present our simplified GSW variant and analysis. In Section 4 we extend this to a homomorphic encryption scheme for symmetric groups. In Section 5.2 we describe and analyze our new bootstrapping algorithm.

Acknowledgments. We thank the anonymous CRYPTO reviewers for their helpful comments, and for pointers to the additional works on branching programs.

2 Preliminaries

For a nonnegative integer n, we let $[n] = \{1, \ldots, n\}$. For an integer modulus q, we let $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ denote the quotient ring of integers modulo q, and $(\mathbb{Z}_q, +)$ its additive group.

2.1 Subgaussian Random Variables

In our constructions it is very convenient to analyze the behavior of "error" terms using the standard notion of *subgaussian* random variables. (For further details and full proofs, see [23].) A real random variable X (or its distribution) is subgaussian with parameter $r > 0$ if for all $t \in \mathbb{R}$, its (scaled) moment-generating function satisfies $\mathbb{E}[\exp(2\pi t X)] \leq \exp(\pi r^2 t^2)$. By a Markov argument, X has Gaussian tails, i.e., for all $t \geq 0$, we have

$$\Pr[|X| \geq t] \leq 2\exp(-\pi t^2/r^2). \tag{1}$$

(If $\mathbb{E}[X] = 0$, then Gaussian tails also imply subgaussianity.) Any B-bounded centered random variable X (i.e., $\mathbb{E}[X] = 0$ and $|X| \leq B$ always) is subgaussian with parameter $B\sqrt{2\pi}$.

Subgaussianity is homogeneous, i.e., X is subgaussian with parameter r, then cX is subgaussian with parameter $c \cdot r$ for any constant $c \geq 0$. Subgaussians also satisfy *Pythagorean additivity*: if X_1 is subgaussian with parameter r_1, and X_2 is subgaussian with parameter r_2 conditioned on *any* value of X_1 (e.g., if X_1 and X_2 are independent), then $X_1 + X_2$ is subgaussian with parameter $\sqrt{r_1^2 + r_2^2}$. By induction this extends to the sum of any finite number of variables, each of which is subgaussian conditioned on any values of the previous ones.

We extend the notion of subgaussianity to vectors: a random real vector $\mathbf{x}$ is subgaussian with parameter r if for all fixed real unit vectors $\mathbf{u}$, the marginal $\langle \mathbf{u}, \mathbf{x} \rangle \in \mathbb{R}$ is subgaussian with parameter r. In particular, it follows directly from the definition that the concatenation of variables or vectors, each of which is subgaussian with common parameter r conditioned on any values of the prior ones, is also subgaussian with parameter r. Homogeneity and Pythagorean additivity clearly extend to subgaussian vectors as well, by linearity.

2.2 Symmetric Groups and $\mathbb{Z}_q$-Embeddings

Here we recall some basic facts about symmetric groups, which can be found in most abstract algebra textbooks, e.g., [16]. Let S_r denote the symmetric group of order r, i.e., the group of permutations (bijections) $\pi\colon \{1,\ldots,r\} \to \{1,\ldots,r\}$ with function composition as the group operation. The group S_r is isomorphic to the multiplicative group of r-by-r *permutation matrices* (i.e., 0-1 matrices with exactly one nonzero element in each row and each column), via the map that associates $\pi \in S_r$ with the permutation matrix $\mathbf{P}_\pi = [\mathbf{e}_{\pi(1)}\ \mathbf{e}_{\pi(2)}\ \cdots\ \mathbf{e}_{\pi(r)}]$, where $\mathbf{e}_i \in \{0,1\}^r$ is the ith standard basis vector. For the remainder of this work we identify permutations with their associated permutation matrices.

The additive cyclic group $(\mathbb{Z}_r, +)$ embeds into the symmetric group S_r via the injective homomorphism that sends the generator $1 \in \mathbb{Z}_r$ to the "cyclic shift" permutation $\pi \in S_r$, defined as $\pi(i) = i+1$ for $1 \le i < r$ and $\pi(r) = 1$.[3] Clearly, this embedding and its inverse can be computed efficiently. Notice also that the permutation matrices in the image of this embedding can be represented more compactly by just their first column, because the remaining columns are just the successive cyclic shifts of this column. Similarly, such permutation matrices can be multiplied in only $O(r^2)$ operations, since we only need to multiply one matrix by the first column of the other.

For our efficient bootstrapping algorithm, we need to efficiently embed a group $(\mathbb{Z}_q, +)$, for some sufficiently large q of our choice, into a symmetric group of order much smaller than q (e.g., polylogarithmic in q). This can be done as follows: suppose that $q = r_1 r_2 \cdots r_t$, where the r_i are pairwise coprime. Then by the Chinese Remainder Theorem, the ring $\mathbb{Z}_q$ is isomorphic to the direct product of rings $\mathbb{Z}_{r_1} \times \mathbb{Z}_{r_2} \times \cdots \times \mathbb{Z}_{r_t}$, and hence their additive groups are isomorphic as well. Combining this with the group embeddings of $(\mathbb{Z}_{r_i}, +)$ into S_{r_i}, we have an (efficient) group embedding from $(\mathbb{Z}_q, +)$ into $S_{r_1} \times S_{r_2} \times \cdots \times S_{r_t}$.[4]

Importantly for our purposes, q can be exponentially large in terms of $\max_i r_i$ above. This can be shown using lower bounds on the second Chebyshev function

$$\psi(x) := \sum_{p^k \le x} \log p = \log\Big(\prod_{p \le x} p^{\lfloor \log_p x \rfloor}\Big),$$

where the first summation is over all prime powers $p^k \le x$, and the second is over all primes $p \le x$; note that $p^{\lfloor \log_p x \rfloor}$ is the largest power of p not exceeding x. Therefore, the product q of all maximal prime powers $r_i = p^{\lfloor \log_p x \rfloor} \le x$ is $\exp(\psi(x))$. Asymptotically, it is known that $\psi(x) = x \pm O(x/\log x)$, and we also have the nonasymptotic bound $\psi(x) \ge 3x/4$ for all $x \ge 7$ [21, Theorem 11]. In summary:

Lemma 2.1. *For all $x \ge 7$, the product of all maximal prime powers $r_i \le x$ is at least* $\exp(3x/4)$.

[3] This is just a special case of Cayley's theorem, which says that any group G embeds into the symmetric group $S_{|G|}$.

[4] The latter group can be seen as a subgroup of S_r for $r = \sum_i r_i$, but it will be more efficient to retain the product structure.

For any given lower bound $q_0 \geq 191 > \exp(21/4)$, we can therefore efficiently find a $q \geq q_0$ whose maximal prime-power divisors are all at most $x = \frac{4}{3} \log q_0 \geq 7$.

3 GSW Cryptosystem

Here we present a variant of the Gentry-Sahai-Waters homomorphic encryption scheme [15] (hereafter called GSW), which we believe is simpler to understand at a technical level. We also give a tighter analysis of its error growth under homomorphic operations.

We first recall some standard background (see, e.g., [18] for further details). For a modulus q, let $\ell = \lceil \log_2 q \rceil$ and define the "gadget" (column) vector $\mathbf{g} = (1, 2, 4, \ldots, 2^{\ell-1}) \in \mathbb{Z}_q^\ell$. Note that the penultimate entry $2^{\ell-2}$ of $\mathbf{g}$ is in the interval $[q/4, q/2) \bmod q$. It will be convenient to use the following randomized "decomposition" function.

Claim (Adapted from [18]). There is a randomized, efficiently computable function $\mathbf{g}^{-1} \colon \mathbb{Z}_q \to \mathbb{Z}^\ell$ such that $\mathbf{x} \leftarrow \mathbf{g}^{-1}(a)$ is subgaussian with parameter $O(1)$, and always satisfies $\langle \mathbf{g}, \mathbf{x} \rangle = a$.

For vectors and matrices over $\mathbb{Z}_q$, define the randomized function $\mathbf{G}^{-1} \colon \mathbb{Z}_q^{n \times m} \to \mathbb{Z}^{n\ell \times m}$ by applying $\mathbf{g}^{-1}$ independently to each entry. Notice that for any $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, if $\mathbf{X} \leftarrow \mathbf{G}^{-1}(\mathbf{A})$ then $\mathbf{X}$ has subgaussian parameter $O(1)$ and

$$\mathbf{G} \cdot \mathbf{X} = \mathbf{A}, \quad \text{where} \quad \mathbf{G} = \mathbf{g}^t \otimes \mathbf{I}_n = \mathrm{diag}(\mathbf{g}^t, \ldots, \mathbf{g}^t) \in \mathbb{Z}_q^{n \times n\ell} \tag{2}$$

is the block matrix with n copies of $\mathbf{g}^t$ as diagonal blocks, and zeros elsewhere.

3.1 Cryptosystem and Homomorphic Operations

The GSW scheme is parameterized by a dimension n, a modulus q with $\ell = \lceil \log_2 q \rceil$, and some error distribution χ over $\mathbb{Z}$ which we assume to be subgaussian. Formally, the message space is the ring of integers $\mathbb{Z}$, though for bootstrapping we only work with ciphertexts encrypting messages in $\{0, 1\} \subset \mathbb{Z}$. The ciphertext space is $\mathcal{C} = \mathbb{Z}_q^{n \times n\ell}$. For simplicity we present just a symmetric-key scheme, which is sufficient for our purposes (it can be converted to a public-key or even attribute-based scheme, as described in [15]).

Our GSW variant differs from the original scheme described in [15] in two main ways:

1. In [15], a ciphertext is a square binary matrix $\mathbf{C} \in \{0, 1\}^{n\ell}$, a secret key is a "structured" mod-q vector $\mathbf{s} \in \mathbb{Z}_q^{n\ell}$ (having large entries), and $\mathbf{s}$ is an "approximate mod-q eigenvector" of $\mathbf{C}$, in the sense that $\mathbf{s}^t \mathbf{C} \approx \mu \mathbf{s}^t \pmod q$, where $\mu \in \mathbb{Z}$ is the message.
   In our variant, a ciphertext is a rectangular mod-q matrix $\mathbf{C} \in \mathbb{Z}_q^{n \times n\ell}$, a secret key is some (unstructured, short) integer vector $\mathbf{s} \in \mathbb{Z}^n$, and $\mathbf{s}^t \mathbf{C} \approx \mu \cdot \mathbf{s}^t \mathbf{G} \pmod q$, i.e., $\mathbf{s}$ and $\mathbf{G}^t \mathbf{s}$ are corresponding left- and right- "approximate singular vectors" of $\mathbf{C}$.

The difference between these two variants turns out to be purely syntactic, in that we can efficiently and "losslessly" switch between them (without needing the secret key). However, we believe that our variant leads to simpler notation and easier-to-understand operations and analysis.

2. The second difference is more substantial: our homomorphic multiplication procedure uses the *randomized* $\mathbf{G}^{-1}(\cdot)$ operation from Claim 3. This yields a few important advantages, such as a very tight and simple error analysis using subgaussianity (see Lemma 3.3), and the ability to *completely re-randomize* the error in a ciphertext (see Corollary 3.4).

We now describe the scheme formally.

GSW.$\mathsf{Gen}()$**:** choose $\bar{\mathbf{s}} \leftarrow \chi^{n-1}$ and output secret key $\mathbf{s} = (\bar{\mathbf{s}}, 1) \in \mathbb{Z}^n$.

GSW.$\mathsf{Enc}((\bar{\mathbf{s}}, 1), \mu \in \mathbb{Z})$**:** choose $\bar{\mathbf{C}} \leftarrow \mathbb{Z}_q^{(n-1)\times n\ell}$ and $\mathbf{e} \leftarrow \chi^m$, let $\mathbf{b}^t = \mathbf{e}^t - \bar{\mathbf{s}}^t\bar{\mathbf{C}} \pmod q$, and output the ciphertext

$$\mathbf{C} = \begin{pmatrix} \bar{\mathbf{C}} \\ \mathbf{b}^t \end{pmatrix} + \mu\mathbf{G} \in \mathcal{C},$$

where $\mathbf{G}$ is as defined in Equation (2). Notice that $\mathbf{s}^t\mathbf{C} = \mathbf{e}^t + \mu \cdot \mathbf{s}^t\mathbf{G} \pmod q$.

GSW.$\mathsf{Dec}(\mathbf{s}, \mathbf{C} \in \mathcal{C})$**:** let $\mathbf{c}$ be the penultimate column of $\mathbf{C}$, and output $\mu = \lfloor \langle \mathbf{s}, \mathbf{c} \rangle \rceil_2$, where $\lfloor \cdot \rceil_2 \colon \mathbb{Z}_q \to \{0, 1\}$ indicates whether its argument is closer modulo q to 0 or to $2^{\ell-2}$ (the penultimate entry of $\mathbf{g}$).[5]

Homomorphic addition is defined as $\mathbf{C}_1 \boxplus \mathbf{C}_2 = \mathbf{C}_1 + \mathbf{C}_2$.

Homomorphic multiplication is defined as $\mathbf{C}_1 \boxdot \mathbf{C}_2 \leftarrow \mathbf{C}_1 \cdot \mathbf{G}^{-1}(\mathbf{C}_2)$, and is *right associative*. Notice that this is a randomized procedure, because $\mathbf{G}^{-1}$ is randomized.

The IND-CPA security of the scheme follows immediately from the assumed hardness of $\mathsf{LWE}_{n-1,q,\chi}$, where the entries of the secret are drawn from the error distribution χ (which is no easier than for a uniformly random secret; see [2, Lemma 2]). This is because a fresh ciphertext is just $\mu\mathbf{G}$ plus a matrix of $n\ell$ independent LWE samples under secret $\bar{\mathbf{s}}$, which are pseudorandom by assumption and hence hide $\mu\mathbf{G}$.

3.2 Analysis

Here we analyze the scheme's correctness and homomorphic operations.

Definition 3.1. *We say that a ciphertext* $\mathbf{C}$ *is* designed *to encrypt message* $\mu \in \mathbb{Z}$ *(under a secret key* $\mathbf{s}$*) if it is a fresh encryption of* μ*, or if* $\mathbf{C} = \mathbf{C}_1 \boxplus \mathbf{C}_2$ *where* $\mathbf{C}_1, \mathbf{C}_2$ *are respectively designed to encrypt* $\mu_1, \mu_2 \in \mathbb{Z}$ *and* $\mu = \mu_1 + \mu_2$*, or similarly for homomorphic multiplication.*

[5] Note that we can decrypt messages in $\mathbb{Z} \cap [-\frac{q}{2}, \frac{q}{2})$, or any other canonical set of representatives of $\mathbb{Z}_q$, by "decoding" $\mathbf{s}^t\mathbf{C}$ to the nearest multiple of $\mathbf{s}^t\mathbf{G}$. The above decryption algorithm will be sufficient for our purposes.

Definition 3.2. *We say that a ciphertext* $\mathbf{C}$ *that is designed to encrypt* $\mu \in \mathbb{Z}$ *(under* $\mathbf{s}$*) has* error *vector* $\mathbf{e}^t \in \mathbb{Z}^{n\ell}$ *if* $\mathbf{s}^t\mathbf{C} - \mu \cdot \mathbf{s}^t\mathbf{G} = \mathbf{e}^t \pmod q$.

For convenience later on, we also say the matrix $\mu\mathbf{G}$ is designed to encrypt μ, and has error $\mathbf{0}$. (This is essentially implied by the above definitions, since $\mu\mathbf{G}$ is indeed a fresh encryption of μ, assuming that zero is in the support of χ.) The next claim on the correctness of decryption follows immediately from the fact that $\mathbf{s} = (\bar{\mathbf{s}}, 1)$ and the penultimate column of $\mathbf{G}$ is $(0, \ldots, 0, 2^{\ell-2})$, where $2^{\ell-2} \in [q/4, q/2) \bmod q$.

Claim. If $\mathbf{C}$ is designed to encrypt some $\mu \in \{0, 1\} \subset \mathbb{Z}$, and has error vector $\mathbf{e}^t$ whose penultimate coordinate has magnitude less than $q/8$, then $\mathrm{GSW.Dec}(\mathbf{s}, \mathbf{C})$ correctly outputs μ.

We now analyze the behavior of the error terms under homomorphic operations.

Lemma 3.3. *Suppose* $\mathbf{C}_1, \mathbf{C}_2$ *are respectively designed to encrypt* $\mu_1, \mu_2 \in \mathbb{Z}$ *and have error vectors* $\mathbf{e}_1^t, \mathbf{e}_2^t$*. Then* $\mathbf{C}_1 \boxplus \mathbf{C}_2$ *has error vector* $\mathbf{e}_1^t + \mathbf{e}_2^t$*, and* $\mathbf{C}_1 \boxdot \mathbf{C}_2$ *has error vector* $\mathbf{e}_1^t\mathbf{X} + \mu_1\mathbf{e}_2^t$*, where* $\mathbf{X} \leftarrow \mathbf{G}^{-1}(\mathbf{C}_2)$ *is the matrix used in the evaluation of* $\boxdot$*. In particular, for* any *values of* $\mathbf{C}_i, \mathbf{e}_i, \mu_i$*, the latter error vector is of the form* $\mathbf{e}^t + \mu_1\mathbf{e}_2^t$*, where the entries of* $\mathbf{e}$ *are independent and subgaussian with parameter* $O(\|\mathbf{e}_1\|)$*.*

Importantly, the error in $\mathbf{C}_1 \boxdot \mathbf{C}_2$ is *quasi-additive* and *asymmetric* with respect to the errors in $\mathbf{C}_1, \mathbf{C}_2$: while the first error vector $\mathbf{e}_1^t$ is multiplied by a short (subgaussian) matrix $\mathbf{X}$, the second error vector $\mathbf{e}_2^t$ is only multiplied by the (scalar) *message* μ_1, which we will ensure remains in $\{0, 1\}$.

Proof. The first claim is immediate, by linearity. For the second claim, because $\mathbf{G} \cdot \mathbf{X} = \mathbf{C}_2$ we have

$$\begin{aligned}
\mathbf{s}^t(\mathbf{C}_1 \boxdot \mathbf{C}_2) &= \mathbf{s}^t\mathbf{C}_1 \cdot \mathbf{X} \\
&= (\mathbf{e}_1^t + \mu_1 \cdot \mathbf{s}^t\mathbf{G})\mathbf{X} \\
&= \mathbf{e}_1^t\mathbf{X} + \mu_1(\mathbf{e}_2^t + \mu_2 \cdot \mathbf{s}^t\mathbf{G}) \\
&= (\mathbf{e}_1^t\mathbf{X} + \mu_1\mathbf{e}_2^t) + \mu_1\mu_2 \cdot \mathbf{s}^t\mathbf{G}.
\end{aligned}$$

As observed in [7], the asymmetric noise growth allows for performing a long chain of homomorphic multiplications while only incurring a polynomial-factor error growth, because $\boxdot$ is defined to be right associative. For convenience of analysis, in such a chain we always include the fixed ciphertext $\mathbf{G}$, which is designed to encrypt $\mu = 1$ and has zero error, as the rightmost ciphertext in the chain. This ensures that the error vector of the output ciphertext is subgaussian and essentially *independent* of the errors in the input ciphertexts (apart from their lengths), which leads to a simpler and tighter analysis. (In [7] a weaker independence guarantee was achieved by a separate "partial re-randomization" procedure, which requires additional public key material.)

Corollary 3.4. *Suppose that* $\mathbf{C}_i$ *for* $i \in [k]$ *are respectively designed to encrypt* $\mu_i \in \{0, \pm 1\}$ *and have error vectors* $\mathbf{e}_i^t$*. Then for* any *fixed values of these variables,*

$$\mathbf{C} \leftarrow \boxed{\cdot}_{i \in [k]} \mathbf{C}_i \ \boxdot\ \mathbf{G} = \mathbf{C}_1 \boxdot (\mathbf{C}_2 \boxdot (\cdots (\mathbf{C}_k \boxdot \mathbf{G}) \cdots))$$

has an error vector whose entries are mutually independent and subgaussian with parameter $O(\|\mathbf{e}\|)$*, where* $\mathbf{e}^t = (\mathbf{e}_1^t, \ldots, \mathbf{e}_k^t) \in \mathbb{Z}^{kn\ell}$ *is the concatenation of the individual error vectors.*

Proof. By Lemma 3.3, the error vector in $\mathbf{C}$ is $\sum_i \mathbf{e}_i^t \mathbf{X}_i$, where each $\mathbf{e}_i^t \mathbf{X}_i$ is a fresh independent vector that has mutually independent coordinates and is subgaussian with parameter $O(\|\mathbf{e}_i\|)$. The claim then follows by Pythagorean additivity.

4 Homomomorphic Encryption for Symmetric Groups

Brakerski and Vaikuntanathan [7] showed how to use the GSW encryption scheme to homomorphically compose permutations of five elements (i.e., to homomorphically compute the group operation in the symmetric group S_5) with small additive noise growth; the use of S_5 comes from its essential role in Barrington's theorem [3]. In [7], the homomorphic composition of permutations is intertwined with the evaluation of a branching program given by Barrington's theorem. Here we give, as a "first-class object," a homomorphic cryptosystem for any symmetric group S_r. The ability to use several different small values of r, along with a homomorphic equality test that we design, will be central to our bootstrapping algorithm.

4.1 Encryption Scheme

We now describe our (symmetric-key) homomorphic encryption scheme for symmetric groups, called HEPerm. Let $\mathcal{C}$ denote the ciphertext space for an appropriate instantiation of the GSW scheme, which we treat as a "black box." A secret key sk for HEPerm is simply a secret key for the GSW scheme.

- HEPerm.$\mathsf{Enc}(sk, \pi \in S_r)$: let $\mathbf{P} = (p_{i,j}) \in \{0,1\}^{r \times r}$ be the permutation matrix associated with π. Output an entry-wise encryption of $\mathbf{P}$, i.e., the ciphertext
$$\mathbf{C} = (c_{i,j}) \in \mathcal{C}^{r \times r}, \text{ where } c_{i,j} \leftarrow \mathsf{Enc}(sk, p_{i,j}).$$

(Decryption follows in the obvious manner.) As with the GSW system, we say that a ciphertext $\mathbf{C} \in \mathcal{C}^{r \times r}$ is *designed* to encrypt a permutation $\pi \in S_r$ (or its permutation matrix $\mathbf{P}_\pi$) if its $\mathcal{C}$-entries are designed to encrypt the corresponding entries of $\mathbf{P}_\pi$. For convenience, we let $\mathbf{J} \in \mathcal{C}^{r \times r}$ denote the ciphertext that encrypts the identity permutation with zero noise, which is built in the expected way from the fixed zero-error GSW ciphertexts that encrypt 0 and 1.

We now show how to homomorphically compute two operations: the standard composition operation for permutations, and an equality test.

Homomorphic composition $\mathbf{C}^\pi \boxdot \mathbf{C}^\sigma$: on ciphertexts $\mathbf{C}^\pi = (c^\pi_{i,j}), \mathbf{C}^\sigma = (c^\sigma_{i,j}) \in \mathcal{C}^{r\times r}$ encrypting permutations $\pi, \sigma \in S_r$ respectively, we compute one encrypting the permutation $\pi \circ \sigma$ by homomorphically evaluating the naïve matrix-multiplication algorithm.That is, output $\mathbf{C} = (c_{i,j}) \in \mathcal{C}^{r\times r}$ where

$$c_{i,j} \leftarrow \boxplus_{\ell\in[r]} (c^\pi_{i,\ell} \boxdot c^\sigma_{\ell,j}) \in \mathcal{C}. \tag{3}$$

Just like $\boxdot$, we define $\boxdot$ to be right associative.

Homomorphic equality test $\mathsf{Eq}?(\mathbf{C}^\pi = (c^\pi_{i,j}), \sigma \in S_r)$**:** given a ciphertext encrypting some permutation $\pi \in S_r$ and a permutation $\sigma \in S_r$ (in the clear), output a ciphertext $c \in \mathcal{C}$ encrypting 1 if $\pi = \sigma$ and 0 otherwise, as

$$c \leftarrow \boxdot_{i\in[r]} c^\pi_{\sigma(i),i} \boxdot g,$$

where $g \in \mathcal{C}$ denotes the fixed zero-error encryption of 1. (Recall that $\boxdot$ is right associative.)

Observe that for the above two operations, the GSW ciphertext(s) in the output are designed to encrypt the appropriate $\{0,1\}$-message. For $\mathsf{Compose}$ this is simply by correctness of the matrix-multiplication algorithm. For $\mathsf{Eq}?$ this is because the output ciphertext is designed to encrypt 1 if and only if every $c_{\sigma(i),i}$ is designed to encrypt 1, which is the case if and only if $\mathbf{C}^\pi$ is in fact designed to encrypt σ. All that remains is to analyze the behavior of the error terms, which we do next.

4.2 Analysis

Recalling that the GSW scheme is parameterized by n and q, denote its space of error vectors by $\mathcal{E} = \mathbb{Z}^m$ where $m = n\lceil \log_2 q \rceil$. The Euclidean norm on $\mathcal{E}^r = \mathbb{Z}^{mr}$ is defined in the expected way. In what follows it is often convenient to consider vectors and matrices over $\mathcal{E}$, i.e., each entry is itself a (row) vector in $\mathcal{E} = \mathbb{Z}^m$, and we switch between $\mathcal{E}^{h\times w}$ and $\mathbb{Z}^{h\times wm}$ as is convenient.

The following lemma describes the behavior of errors under the homomorphic composition operation $\boxdot$. Note that working with vectors and matrices over $\mathcal{E}$ lets us write a statement that is syntactically very similar to the one from Lemma 3.3, with a very similar proof.

Lemma 4.1. *Let $\mathbf{C}^\pi, \mathbf{C}^\sigma \in \mathcal{C}^{r\times r}$ respectively be designed to encrypt permutation matrices $\mathbf{P}^\pi, \mathbf{P}^\sigma \in \{0,1\}^{r\times r}$ with error matrices $\mathbf{E}^\pi, \mathbf{E}^\sigma \in \mathcal{E}^{r\times r}$. Then for* any *fixed values of these variables, $\mathbf{C}^\pi \boxdot \mathbf{C}^\sigma$ has error matrix $\mathbf{E} + \mathbf{P}^\pi \cdot \mathbf{E}^\sigma \in \mathcal{E}^{r\times r}$, where the $\mathbb{Z}$-entries of $\mathbf{E}$ are mutually independent, and those in its ith row are subgaussian with parameter $O(\|\mathbf{e}^\pi_i\|)$, where $\mathbf{e}^\pi_i$ is the ith row of $\mathbf{E}^\pi$.*

Proof. Let $\mathbf{C} \leftarrow \mathbf{C}^\pi \boxdot \mathbf{C}^\sigma$. It suffices to show that for all i, j, its (i,j)th entry $c_{i,j} \in \mathcal{C}$ has error

$$e_{i,j} + e^\sigma_{\pi^{-1}(i),j} \in \mathcal{E} = \mathbb{Z}^m,$$

where all the $\mathbb{Z}$-entries of all the $e_{i,j} \in \mathbb{Z}^m$ are mutually independent and subgaussian with parameter $O(\|\mathbf{e}_i^\pi\|)$, and $e_{\ell,j}^\sigma$ is the (ℓ, j)th entry of $\mathbf{E}^\sigma$. This follows directly from Equation (3) and Lemma 3.3: the error in each ciphertext $c_{i,\ell}^\pi \boxdot c_{\ell,j}^\sigma$ is $p_{i,\ell}^\pi \cdot e_{\ell,j}^\sigma$ plus a fresh vector whose entries are independent and subgaussian with parameter $O(\|e_{i,\ell}^\pi\|)$. Since $p_{i,\ell}^\pi = 1$ for $\ell = \pi^{-1}(i)$ and 0 otherwise, the claim follows by Pythagorean additivity of independent subgaussians.

Similarly to a multiplication chain of GSW ciphertexts, we can perform a (right-associative) chain of compositions while incurring only small error growth. For convenience of analysis, we always include the fixed zero-error ciphertext $\mathbf{J} \in \mathcal{C}^{r \times r}$ (which encrypts the identity permutation) as the rightmost ciphertext in the chain. The following corollary follows directly from Lemma 4.1 in the same way that Corollary 3.4 follows from Lemma 3.3.

Corollary 4.2. *Suppose that $\mathbf{C}_i \in \mathcal{C}^{r \times r}$ for $i \in [k]$ are respectively designed to encrypt permutation matrices $\mathbf{P}_i \in \{0,1\}^{r \times r}$ and have error matrices $\mathbf{E}_i \in \mathcal{E}^{r \times r}$. Then for* any *fixed values of these variables,*

$$\mathbf{C} \leftarrow \boxed{\bigcirc}_{i \in [k]} \mathbf{C}_i \boxdot \mathbf{J} = \mathbf{C}_1 \boxdot (\mathbf{C}_2 \boxdot (\cdots (\mathbf{C}_k \boxdot \mathbf{J}) \cdots))$$

has an error matrix whose $\mathbb{Z}$-entries are mutually independent, and those in its ith row are subgaussian with parameter $O(\|\mathbf{e}_i\|)$, where $\mathbf{e}_i^t \in \mathcal{E}^{kr}$ is the ith row of the concatenated error matrices $[\mathbf{E}_1 \mid \cdots \mid \mathbf{E}_k]$.

Finally, since the Eq? procedure simply performs a chain of (right-associative) multiplications of GSW ciphertexts, Corollary 3.4 applies.

4.3 Optimizations for $\mathbb{Z}_r$ Embeddings

For bootstrapping, we use the above scheme only to encrypt elements in the cyclic subgroup $C_r \subseteq S_r$ that embeds the additive group $(\mathbb{Z}_r, +)$. As described in the preliminaries, an element $\pi \in C_r$ can be represented more compactly as an indicator (column) vector $\mathbf{p} \in \{0,1\}^r$ (rather than a matrix in $\{0,1\}^{r \times r}$), and its associated permutation matrix $\mathbf{P}_\pi$ is made up of the r cyclic rotations of $\mathbf{p}$. In addition, the composition of two permutations represented in this way as $\mathbf{p}, \mathbf{q}$ is given by the matrix-vector product $\mathbf{P}_\pi \cdot \mathbf{q}$, which may be computed in $O(r^2)$ operations, rather than $O(r^3)$ as in the general case. All of this translates directly to *encrypted* permutations in the expected way, i.e., ciphertexts are entry-wise encryptions in $\mathcal{C}^r$ of indicator vectors, etc.

Similarly, the equality test Eq? can be performed more efficiently when we restrict to the subgroup C_r: given r ciphertexts encrypting the entries of an indicator vector in $\{0,1\}^r$ and an $s \in \mathbb{Z}_r$, just output the ciphertext in the position corresponding to s.

Since our bootstrapping scheme uses $\mathbb{Z}_r$ embeddings only for $r = O(\log \lambda)$, these optimizations lead to polylogarithmic factor improvements in runtime and error, but no more.

5 Bootstrapping

We now describe our bootstrapping procedure.

5.1 Specification and Usage

We start by specifying the abstract preconditions and output guarantees of our bootstrapping algorithm, and describe how to use it (with some additional pre- and post-processing) to bootstrap known LWE-based encryption schemes.

The scheme to be bootstrapped must have *binary* ciphertexts in $\{0,1\}^d$ and secret keys in $\mathbb{Z}_q^d$ for some dimension d and modulus q that should be made as small as possible ($q, d = \tilde{O}(\lambda)$ are possible), and a decryption function of the form $\mathsf{Dec}_{\mathbf{s}}(\mathbf{c}) = f(\langle \mathbf{s}, \mathbf{c} \rangle) \in \{0,1\}$ for some arbitrary function $f\colon \mathbb{Z}_q \to \{0,1\}$. We rely on an appropriate instantiation of the GSW cryptosystem, as described in further detail in Section 5.2 below.

$\mathsf{BootGen}(\mathbf{s} \in \mathbb{Z}_q^d, sk)$ takes as input a secret key vector $\mathbf{s} \in \mathbb{Z}_q^d$ from the scheme to be bootstrapped, and a secret key sk for GSW. It outputs a bootstrapping key bk, which appropriately encrypts $\mathbf{s}$ under sk.

$\mathsf{Bootstrap}(bk, \mathbf{c} \in \{0,1\}^d)$ takes as input the bootstrapping key bk and a ciphertext vector $\mathbf{c} \in \{0,1\}^d$ (which decrypts under the secret key $\mathbf{s}$). It outputs a GSW ciphertext which decrypts (under sk) to the same bit as $\mathbf{c}$ does (under $\mathbf{s}$), but with less error.

Pre- and post-processing. We can bootstrap all known LWE-based bit-encryption schemes using the above algorithms as follows. In all LWE-based encryption schemes, decryption can be expressed as a "rounded inner product" $\lfloor \langle \mathbf{s}, \mathbf{c} \rangle \rceil_2$ for some appropriate rounding function $\lfloor \cdot \rceil_2\colon \mathbb{Z}_q \to \{0,1\}$, as required. Note that a GSW ciphertext can trivially be put in this form by just taking its penultimate column (see GSW.Dec in Section 3.1). As for the other conditions we need (binary ciphertexts and small d, q), LWE encryption schemes are not always presented in a way that fulfills them, but fortunately there are standard transformations that do so, as we now describe. (See [6, 5] for further details.)

First, since we do not need to perform any further homomorphic operations on the ciphertext, we can use dimension- and modulus-reduction [6] to get a ciphertext $\bar{\mathbf{c}}$ (over $\mathbb{Z}_q$) of dimension $\tilde{O}(\lambda)$ and modulus $q = \tilde{O}(\lambda)$, while preserving correct decryption. These steps can be implemented with 2^λ security under conventional lattice assumptions.[6] Then, we can obtain a binary ciphertext $\mathbf{c}$ using "bit decomposition:" let $\mathbf{G}$ be as defined in Section 3, and for the ciphertext $\bar{\mathbf{c}}$ over $\mathbb{Z}_q$ under secret key $\bar{\mathbf{s}}$, let $\mathbf{c}$ be a $\{0,1\}$-vector such that $\mathbf{G}\mathbf{c} = \bar{\mathbf{c}}$, and let $\mathbf{s} = \mathbf{G}^t \bar{\mathbf{s}}$ so that $\langle \mathbf{s}, \mathbf{c} \rangle = \langle \bar{\mathbf{s}}, \bar{\mathbf{c}} \rangle \in \mathbb{Z}_q$. (The secret key $\mathbf{s}$ is therefore the one we need to provide to $\mathsf{BootGen}$.)

[6] To make the modulus quasi-linear, we need to use randomized (subgaussian) rounding in the modulus-reduction step.

After bootstrapping, the output is a GSW ciphertext $\mathbf{C}$ encrypted under sk (which is just an integer vector). If desired, we can convert this ciphertext back to one for the original LWE cryptosystem, simply by taking the penultimate column of $\mathbf{C}$. We can also key-switch from sk back to the original secret key $\mathbf{s}$. (As usual in bootstrapping, going "full circle" in this way requires an appropriate circular security assumption.)

5.2 Procedures

Our algorithms rely on instantiations of GSW and HEPerm with parameters n, Q, χ. Importantly, the ciphertext modulus Q is *not* the modulus q of the scheme we are bootstrapping, but rather some $Q \gg q$ that is sufficiently larger than the error in $\mathsf{Bootstrap}$'s output ciphertext. Let $\mathcal{C}$ denote the GSW ciphertext space.

Our procedures need q to be of the form $q = \prod_{i \in [t]} r_i$ where the r_i are small and powers of distinct primes (and hence pairwise coprime). Specifically, using Lemma 2.1 we can choose $q = \tilde{O}(\lambda)$ to be large enough by letting it be the product of all maximal prime-powers r_i that are bounded by $O(\log \lambda)$, of which there are $t = O(\log \lambda / \log\log \lambda)$. Let ϕ be the group embedding of $(\mathbb{Z}_q, +) \cong (\mathbb{Z}_{r_1} \times \cdots \times \mathbb{Z}_{r_t}, +)$ into $S = S_{r_1} \times \cdots \times S_{r_t}$ described in Section 2.2, and let ϕ_i denote the ith component of this embedding, i.e., the one from $\mathbb{Z}_q$ into S_{r_i}.

$\mathsf{BootGen}(\mathbf{s} \in \mathbb{Z}_q^d, sk)$**:** given secret key $\mathbf{s} \in \mathbb{Z}_q^d$ for the scheme to be bootstrapped and a secret key sk for HEPerm, embed each coordinate $s_j \in \mathbb{Z}_q$ of $\mathbf{s}$ as $\phi(s_j) \in S$ and encrypt the components under HEPerm. That is, generate and output the bootstrapping key

$$bk = \{\mathbf{C}_{i,j} \leftarrow \text{HEPerm.}\mathsf{Enc}(sk, \phi_i(s_j)) : i \in [t], j \in [d]\}.$$

Recalling that we are working with embeddings of $\mathbb{Z}_{r_i}$, each $\mathbf{C}_{i,j} \in \mathcal{C}^{r_i}$ can be represented as a tuple of r_i GSW ciphertexts encrypting an indicator vector (see Section 4.3). Because $t, r_i = O(\log \lambda)$ and $d = \tilde{O}(\lambda)$, the bootstrapping key consists of $\tilde{O}(\lambda)$ GSW ciphertexts.

$\mathsf{Bootstrap}(bk, \mathbf{c} \in \{0,1\}^d)$**:** given a binary ciphertext $\mathbf{c} \in \{0,1\}^d$, do the following:

Inner Product: Homomorphically compute an encryption of

$$v = \langle \mathbf{s}, \mathbf{c} \rangle = \sum_{j \,:\, c_j = 1} s_j \in \mathbb{Z}_q$$

using the encryptions of the $s_j \in \mathbb{Z}_q$ as embedded into the permutation group S, via a chain of compositions. Formally, for each $i \in [t]$ compute (recalling that $\boxdot$ is right associative, and $\mathbf{J}$ is the fixed HEPerm encryption of the identity permutation)

$$\mathbf{C}_i \leftarrow \boxed{\circ}_{j \text{ s.t. } c_j = 1} \mathbf{C}_{i,j} \boxdot \mathbf{J}. \tag{4}$$

Again, because we are working with embeddings of $\mathbb{Z}_{r_i}$, each $\mathbf{C}_i \in \mathcal{C}^{r_i}$.

Round: Homomorphically map $v \in \mathbb{Z}_q$ to $f(v) \in \mathbb{Z}_2 = \{0,1\}$: for each $x \in \mathbb{Z}_q$ such that $f(x) = 1$, homomorphically test whether $v \stackrel{?}{=} x$ by homomorphically multiplying the GSW ciphertexts resulting from all the equality tests $v \stackrel{?}{=} x \pmod{r_i}$. Then homomorphically sum the results of all the $v \stackrel{?}{=} x$ tests.

Formally, compute and output the GSW ciphertext (recalling that $\boxdot$ is right associative, and $\mathbf{G}$ is the fixed GSW encryption of 1)

$$\mathbf{C} \leftarrow \boxplus_{x \in \mathbb{Z}_q \text{ s.t. } f(x)=1} \left(\boxdot_{i \in [t]} \mathsf{Eq?}(\mathbf{C}_i, \phi_i(x)) \boxdot \mathbf{G} \right). \tag{5}$$

Note that since we are working with embeddings of $\mathbb{Z}_{r_i}$, each $\mathsf{Eq?}(\mathbf{C}_i, \phi_i(x))$ is just some GSW ciphertext component of $\mathbf{C}_i \in \mathcal{C}^{r_i}$ (see Section 4.3).

Because $t, r_i = O(\log \lambda)$ and $d = \tilde{O}(\lambda)$ and by Equations (4) and (5), $\mathsf{Bootstrap}$ performs $\tilde{O}(\lambda)$ homomorphic multiplications and additions on GSW ciphertexts.

5.3 Analysis

The following is our main theorem. The proof is deferred to the full version due to space limitations.

Theorem 5.1. *The above bootstrapping scheme can be instantiated to be correct (with overwhelming probability) and secure assuming that the decisional Shortest Vector Problem (GapSVP) and Shortest Independent Vectors Problem (SIVP) are (quantumly) hard to approximate in the worst case to within $\tilde{O}(n^2\lambda)$ factors on n-dimensional lattices.*

Because all known (quantum) algorithms for poly(n)-factor approximations to GapSVP and SIVP on n-dimensional lattices take $2^{\Omega(n)}$ time, for 2^λ hardness we can take $n = \Theta(\lambda)$, yielding a final approximation factor of $\tilde{O}(n^3)$. This comes quite close to the $O(n^{3/2+\epsilon})$ factors obtained in [7], but *without* any expensive "dimension leveraging:" we use GSW ciphertexts of dimension only $n = O(\lambda)$, rather than some large polynomial in λ. Alternatively, at the cost of a larger dimension $n = \lambda^{1/\epsilon}$, but without using the successive dimension-reduction procedure from [7], we can obtain factors as small as $\tilde{O}(n^{2+\epsilon})$ for any constant $\epsilon > 0$.

References

[1] Alperin-Sheriff, J., Peikert, C.: Practical bootstrapping in quasilinear time. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 1–20. Springer, Heidelberg (2013)

[2] Applebaum, B., Cash, D., Peikert, C., Sahai, A.: Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 595–618. Springer, Heidelberg (2009)

[3] Barrington, D.A.M.: Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. In: STOC, pp. 1–5 (1986)
[4] Brakerski, Z., Gentry, C., Vaikuntanathan, V. (Leveled) fully homomorphic encryption without bootstrapping. In: ITCS, pp. 309–325 (2012)
[5] Brakerski, Z., Langlois, A., Peikert, C., Regev, O., Stehlé, D.: Classical hardness of learning with errors. In: STOC, pp. 575–584 (2013)
[6] Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. In: FOCS, pp. 97–106 (2011)
[7] Brakerski, Z., Vaikuntanathan, V.: Lattice-based FHE as secure as PKE. In: ITCS, p. 1 (2014)
[8] Cai, J.-Y., Lipton, R.J.: Subquadratic simulations of circuits by branching programs. In: 2013 IEEE 54th Annual Symposium on Foundations of Computer Science, pp. 568–573 (1989)
[9] Cleve, R.: Towards optimal simulations of formulas by bounded-width programs. Computational Complexity 1(1), 91–105 (1991)
[10] Gentry, C.: A fully homomorphic encryption scheme. PhD thesis, Stanford University (2009), http://crypto.stanford.edu/craig
[11] Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC, pp. 169–178 (2009)
[12] Gentry, C., Halevi, S.: Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In: FOCS, pp. 107–109 (2011)
[13] Gentry, C., Halevi, S., Smart, N.P.: Better bootstrapping in fully homomorphic encryption. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 1–16. Springer, Heidelberg (2012)
[14] Gentry, C., Halevi, S., Smart, N.P.: Fully homomorphic encryption with polylog overhead. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 465–482. Springer, Heidelberg (2012)
[15] Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 75–92. Springer, Heidelberg (2013)
[16] Jacobson, N.: Basic Algebra I. Dover Publications (2012)
[17] Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. Journal of the ACM 60(6), 43:1–43:35 (2013); Preliminary version in Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (2010)
[18] Micciancio, D., Peikert, C.: Trapdoors for lattices: Simpler, tighter, faster, smaller. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 700–718. Springer, Heidelberg (2012)
[19] Peikert, C.: Public-key cryptosystems from the worst-case shortest vector problem. In: STOC 2009, pp. 333–342 (2009)
[20] Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. J. ACM 56(6), 1–40 (2009); Preliminary version in STOC 2005
[21] Schoenfeld, L.: Sharper bounds for the Chebyshev functions $\theta(x)$ and $\psi(x)$. ii. Mathematics of Computation 30(134), 337–360 (1976)
[22] Sinha, R.K.: Some topics in parallel computation and branching programs. PhD thesis, University of Washington (1995)
[23] Vershynin, R.: Compressed Sensing, Theory and Applications, ch. 5, pp. 210–268. Cambridge University Press (2012), http://www-personal.umich.edu/~romanv/papers/non-asymptotic-rmt-plain.pdf