



Compressible FHE with Applications to PIR

Craig Gentry^(✉) and Shai Halevi^(✉)

Algorand Foundation, New-York City, NY, USA
cbgentry@gmail.com, shaih@alum.mit.edu

Abstract. Homomorphic encryption (HE) is often viewed as impractical, both in communication and computation. Here we provide an additively homomorphic encryption scheme based on (ring) LWE with nearly optimal rate ($1 - \epsilon$ for any $\epsilon > 0$). Moreover, we describe how to compress many Gentry-Sahai-Waters (GSW) ciphertexts (e.g., ciphertexts that may have come from a homomorphic evaluation) into (fewer) high-rate ciphertexts.

Using our high-rate HE scheme, we are able for the first time to describe a single-server private information retrieval (PIR) scheme with sufficiently low computational overhead so as to be practical for large databases. Single-server PIR inherently requires the server to perform at least one bit operation per database bit, and we describe a rate-(4/9) scheme with computation which is not so much worse than this inherent lower bound. In fact it is probably less than whole-database AES encryption – specifically about $2.3 \bmod q$ multiplication per database byte, where q is about 50 to 60 bits. Asymptotically, the computational overhead of our PIR scheme is $\tilde{O}(\log \log \lambda + \log \log \log N)$, where λ is the security parameter and N is the number of database files, which are assumed to be sufficiently large.

1 Introduction

How bandwidth efficient can (fully) homomorphic encryption ((F)HE) be? While it is easy to encrypt messages with almost no loss in bandwidth, the same is generally not true for homomorphic encryption: Encrypted ciphertexts in contemporary HE schemes tend to be significantly larger than the plaintext that they encrypt, at least by a significant constant factor and often much more.

Beyond the fundamental theoretical interest in the bandwidth limits of FHE, a homomorphic scheme with high rate has several applications. Perhaps the most obvious is for private information retrieval (PIR), where bandwidth is of the essence. While HE can clearly be used to implement PIR, even the best PIR implementation so far (such as [1, 3]) are still quite far from being able to support large databases, mostly because the large expansion factor of contemporary HE schemes. Another application can be found in the work of Badrinarayanan

This work was done while the authors were in IBM Research.

et al. [6], who showed that compressible (additive) homomorphic encryption with rate better than $1/2$ can be used for a high-rate oblivious transfer, which in turn can be used for various purposes in the context of secure computation. Alas, prior to our work the only instantiation of high rate homomorphic encryption was the Damgård-Jurik cryptosystem [14], which however is (a) only additively homomorphic, (b) rather expensive, and (c) insecure against quantum computers.

In this work we remedy this situation, devising the first *compressible* fully homomorphic encryption scheme, and showing how to use it to get efficient PIR. Namely, we describe an (F)HE scheme whose evaluated ciphertexts can be publicly compressed until they are roughly the same size as the plaintext that they encrypt. Our compressible scheme can take “bloated” evaluated ciphertexts of the GSW cryptosystem [17], and cram them into high-rate matrix-encrypting matrix-ciphertexts. The ratio of the aggregate plaintext size to the aggregate ciphertext size can be $1 - \epsilon$ for any ϵ (assuming the aggregate plaintext is sufficiently large, proportional to $1/\epsilon^3$). The compressed ciphertexts are no longer GSW ciphertexts. However, they still have sufficient structure to allow additive homomorphism, and multiplication by encryption of small scalars, all while remaining compressed.¹ Just like GSW, the security of our scheme is based on the learning with errors assumption [31] or its ring variant [26]. (Also circular security assumption to get *fully* homomorphic encryption.)

We note that a compressible fully homomorphic encryption easily yields an end-to-end rate-efficient FHE: Freshly encrypted ciphertexts are immediately compressed during encryption,² then “decompressed” using bootstrapping before any processing, and finally compressed again before decryption. The resulting scheme has compressed ciphertexts at any time, which are only temporarily expanded while they are being processed.

1.1 Applications to PIR

We describe many optimizations to the basic scheme, yielding a single-server private information retrieval scheme with low communication overhead, while at the same time being computationally efficient. Asymptotically, the computational overhead is $\tilde{O}(\log \log \lambda + \log \log \log N)$, where λ is the security parameter and N is the number of database files, which are assumed to be sufficiently large.

While we did not implement our PIR scheme, we explain in detail why we estimate that it should be not only theoretically efficient but also practically fast. Specifically, we can get a rate $4/9$ single-server PIR scheme,³ in which the server’s amortized work is only 2.3 single-precision modular multiplications for

¹ Of course, these operations increase the noisiness of the ciphertexts somewhat.

² One could even use hybrid encryption, where fresh ciphertexts are generated using, e.g., AES-CTR, and the AES key is sent along encrypted under the FHE.

³ The rate can be made arbitrarily close to one without affecting the asymptotic efficiency, but the concrete parameters of this solution are not appealing. See discussion at the end of Sect. 5.

every byte in the database. For a comparison point, the trivial PIR solution of sending the entire database will have to at least encrypt the whole database (for communication security), hence incurring a cost of an AES block encryption per 16 database bytes, which is surely more work than what our scheme does. Thus, contra Sion-Carbutar [33], PIR is finally more efficient than the trivial solution not only in terms of communication, but also in terms of computation.

Those accustomed to thinking of (R)LWE-based homomorphic encryption as impractical may find the low computational overhead of our PIR scheme hard to believe. However, RLWE-based HE – in particular, the GSW scheme with our adaptations – really shines in the PIR setting for a few reasons. First, the noise in GSW ciphertexts grows only additively with the degree when the messages multiplied from the left are in $\{0, 1\}$. (The receiver’s GSW ciphertexts will encrypt the bits of its target index.) Second, even though we obviously need to do $\Omega(N)$ ciphertext operations for a database with N files, we can ensure that the noise grows only proportionally to $\log N$ (so its bit size only grows with $\log \log N$). The small noise growth allows our PIR scheme to use a small RLWE modulus $q = \tilde{O}(\log N + \lambda)$ that in practice is not much larger than one would use in a basic RLWE-based PKE scheme. Third, we can exploit the recursive/hierarchical nature of the classic approach to single-server PIR [23, 35] to hide the more expensive steps of RLWE-based homomorphic evaluation, namely polynomial FFTs (and less importantly, CRT lifting). In the classical hierarchical approach to PIR, the computationally dominant step is the first step, where we project the effective database size from $N = N_1 \times \dots \times N_d$ down to N/N_1 . To maximize the efficiency of this first step, we can preprocess the polynomials of the database so that they are already in evaluation representation, thereby avoiding polynomial FFTs and allowing each $(\log q)$ -bit block of the database to be “absorbed” into an encrypted query using a small constant number of mod- q multiplications.⁴ Therefore, the computational overhead of the first step boils down to just the overhead of multiplying integers modulo q , where this overhead is $\tilde{O}(\log \log q)$, where (again) q is quite small. After the first step of PIR, GSW-esque homomorphic evaluation requires converting between coefficient and evaluation representation of polynomials, but this will not significantly impact the overhead of our PIR scheme, as the effective database is already much smaller (at most N/N_1), where we will take $N_1 = \tilde{\Theta}(\log N + \lambda)$.

1.2 Related Work

Ciphertext Compression. Ciphertext compression has always had obvious appeal in the public-key setting (and even sometimes in the symmetric key context, e.g., [22]). In the context of (F)HE, one can view “ciphertext packing” [8, 9, 30, 34], where each ciphertext encrypts not one but an array of plaintext elements, as a form of compression. Other prior works included a “post-evaluation” ciphertext compression techniques, such as the work of van Dijk et al. [36] for

⁴ In the first step, the server generates N_1 ciphertexts from the client’s $\log N_1$ ciphertexts, which includes FFTs, but their amortized cost is insignificant when $N_1 \ll N$.

integer-based HE, and the work of Hohenberger et al. for attribute-based encryption [18]. However, the rate achieved there is still low, and in fact no scheme prior to our work was able to break the rate-1/2 barrier. (Hence for example no LWE-based scheme could be used for the high-rate OT application of Badrinarayanan et al. [6].)

The only prior cryptosystem with homomorphic properties that we know of with rate better than 1/2 is due to Damgård and Jurik [14]. They described an extension of the Paillier cryptosystem [29] that allows rate- $(1 - o(1))$ encryption with additive homomorphism: In particular, a $\text{mod-}N^s$ plaintext can be encrypted inside a $\text{mod-}N^{s+1}$ ciphertext for an RSA modulus N and an arbitrary exponent $s \geq 1$.

Finally, a concurrent work by Döttling et al. [15] and follow-up work by Brakerski et al. [7] also achieves compressible variants of HE/FHE. The former work achieves only weaker homomorphism but under a wide variety of hardness assumptions, while the latter achieves FHE under LWE. The constructions in those works are more general than ours, but they are unlikely to yield practical schemes for applications such as PIR.

Private Information Retrieval. Private information retrieval (PIR) [12] lets a client obtain the N -th bit (or file) from a database while keeping its target index $i \in [N]$ hidden from the server(s). To rule out a trivial protocol where the server transmits the entire database to the client, it is required that the total communication is sublinear in N . Chor et al. provided constructions with multiple servers, and later Kushilevitz and Ostrovsky [23] showed that PIR is possible even with a single server under computational assumptions.

Kiayias et al. [21] (see also [25]) gave the first single-server PIR scheme with rate $(1 - o(1))$, based on Damgård-Jurik [14]. However, Damgård-Jurik is computationally too expensive to be used in practice for large-scale PIR [28, 33], at a minimum, PIR using Damgård-Jurik requires the server to compute a $\text{mod-}N$ multiplication per bit of the database, where N has 2048 or more bits. The papers [21, 25] expressly call for an underlying encryption scheme to replace Damgård-Jurik to make their rate-optimal PIR schemes computationally less expensive.

In terms of computation, the state-of-the-art PIR scheme is XPIR by Aguilar-Melchor et al. [1], with further optimizations in the SealPIR work of Angel et al. [3]. This scheme is based on RLWE and features many clever optimizations, but Angel et al. commented that even with their optimizations “supporting large databases remains out of reach.” Concretely, the SealPIR results from [3, Fig. 9] indicate server workload of about twenty cycles per database byte, for a rate of roughly 1/1000. In contrast, our construction yields rate close to 1/2, and the server work-load is roughly 2.3 single-precision modular multiplication per byte (this should be less than 20 cycles).

Organization. Some background information regarding LWE and the GSW scheme is provided in Sect. 2. In Sect. 3 we define compressible HE, in Sect. 4 we describe our compressible (F)HE scheme, and in Sect. 5, we describe our PIR scheme.

2 Background on Gadget Matrices, LWE, PVW and GSW

Gadget Matrices. Many lattice cryptosystems (including GSW [17]) use a rectangular *gadget matrix* [27], $G \in R_q^{n \times m}$ to add redundancy. For a matrix C of dimension $n \times c$ we denote by $G^{-1}(C)$ a matrix of dimension $m \times c$ with small coefficients such that $G \cdot (G^{-1}(C)) = C \pmod{q}$. Below we also use the convention that $G^{-1}(C)$ is always a *full rank matrix* over the rationals⁵. In particular we denote by $G^{-1}(0)$ a matrix M with small entries and full rank over the rationals, such that $G \cdot M = 0 \pmod{q}$ (so clearly M does not have full rank modulo q). Often G is set to be $I_{n_1} \otimes \mathbf{g}$ where \mathbf{g} is the vector $(1, 2, \dots, 2^{\lceil \log q \rceil})$ – that is, $m = n_1 \lceil \log q \rceil$ and G 's rows consists of shifts of the vector \mathbf{g} . In this case, one can efficiently find a suitable $G^{-1}(C)$ that has coefficients in $\{0, 1\}$. More generally with $\mathbf{g} = (1, B, \dots, B^{\lceil \log_B q \rceil})$, $G^{-1}(C)$ has coefficients in $[\pm B/2]$.

(Ring) Learning With Errors (LWE). Security of many lattice cryptosystems is based on the hardness of the decision (ring) learning with errors (R)LWE problem [26,31]. LWE uses the ring of integers $R = \mathbb{Z}$, while RLWE typically uses the ring of integers R of a cyclotomic field. A “yes” instance of the (R)LWE problem for modulus q , dimension k , and noise distribution χ over R consists of many uniform $\mathbf{a}_i \in R_q^k$ together with the values $b_i := \langle \mathbf{s}, \mathbf{a}_i \rangle + e_i \in R_q$ where \mathbf{s} is a fixed secret vector and $e_i \leftarrow \chi$. In a “no” instance, both the \mathbf{a}_i 's and b_i 's are uniform. The decision (R)LWE assumption is that the two distributions are computationally indistinguishable – i.e., that “yes” instances are pseudorandom. Typically, χ is such that $\|e_i\|_\infty < \alpha$ for some size bound α with probability overwhelming in the security parameter λ . The security parameter also lower bounds the ring size and/or the dimension k , and the ratio α/q .

LWE with Matrix Secrets. An LWE instance may (more generally) be associated to a secret *matrix* S' , and one can prove via a hybrid argument that breaking the matrix version of LWE is as hard as breaking conventional LWE. In this version, a “yes” instance consists of a uniform matrix A and $B = S'A + E$. Let us give dimensions to these matrices: S' is $n_0 \times k$, A is $k \times m$, B and E are $n_0 \times m$. (See Fig. 1 for an illustration of these matrices.) Set $n_1 = n_0 + k$. Set $S = [S'|I] \in R_q^{n_0 \times n_1}$ and P to be the matrix with $-A$ on top of B . Then $SP = E \pmod{q}$. The LWE assumption (matrix version) says that this P is pseudorandom.

Regev and PVW Encryption. Recall that in the Regev cryptosystem, a bit $\sigma \in \{0, 1\}$ is encrypted relative to a secret-key vector $\mathbf{s} \in \mathbb{Z}_q^{n+1}$ (with 1 in the last coordinate), as a vector $\mathbf{c} \in \mathbb{Z}_q^{n+1}$ such that $\langle \mathbf{s}, \mathbf{c} \rangle = \lceil q/2 \rceil \cdot \sigma + e \pmod{q}$, with $|e| < q/4$. More generally the plaintext space can be extended to R_p for some $p < q$, where a scalar $\sigma \in R_p$ is encrypted as a vector $\mathbf{c} \in R_q^{n+1}$ such that $\langle \mathbf{s}, \mathbf{c} \rangle = \lceil q/p \rceil \cdot \sigma + e \pmod{q}$, with $\|e\|_\infty < q/2p$. (There is also a public key in this cryptosystem and an encryption procedure, but those are not relevant to our construction.)

⁵ More generally, if the matrices are defined over some ring R then we require full rank over the field of fractions for that ring.

Peikert et al. described in [30] a batched variant of this cryptosystem (called PVW), where the plaintext is a vector $\sigma \in R_p^k$, the secret key is a matrix $S = (S'|I) \in R_q^{n \times (n+k)}$ and the encryption of σ is a vector $c \in R_q^{n+k}$ such that $S \cdot c = \lceil q/p \rceil \cdot \sigma + e$ with $\|e\|_\infty < q/2p$. For notational purposes, it will be convenient to use a “matrix variant” of PVW, simply encrypting many vectors and putting them in a matrix. Here the plaintext is a matrix $\Sigma \in R_p^{k \times m}$ (for some m), and the encryption of Σ is a matrix $C \in R_q^{(n+k) \times m}$ such that $SC = \lceil q/p \rceil \cdot \Sigma + E$ with $\|E\|_\infty < q/2p$.

The Regev and PVW cryptosystems are additively homomorphic, supporting addition and multiplication by small scalars, as long as the noise remains small enough. The information rate of the PVW cryptosystem is $\frac{|q|}{|p|} \cdot \frac{k}{n+k}$, which can be made very close to one if we use $k \gg n$ and $q \approx p^{1+\epsilon}$. Indeed this forms the basis for one variant of our compressible (F)HE construction.

GSW Encryption with Matrix Secret Keys. We use (a slight variant of) the GSW cryptosystem of Gentry et al. [17], based on LWE with matrix secret as above. Namely the secret key is a matrix S and the public key is a pseudorandom matrix P such that $SP = E \pmod q$ for a low-norm noise matrix E .

The plaintext space of GSW are (small) scalars. To encrypt $\sigma \in R_q$ under GSW, the encrypter chooses a random $m \times m$ matrix X whose entries have small norm, and outputs $C = \sigma \cdot G + P \cdot X \in R_q^{n_1 \times m}$ (operations modulo q). To decrypt, one computes

$$S \cdot C = \sigma \cdot S \cdot G + S \cdot P \cdot X = \sigma \cdot S \cdot G + E' \pmod q, \quad (1)$$

where $E' = E \cdot X$ has small norm. Assuming E' has coefficients bounded by an appropriate β , then $E' \cdot G^{-1}(0)$ will have entries too small to wrap modulo q , allowing the decrypter to recover E' (since $G^{-1}(0)$ is invertible) and hence recover $\sigma \cdot S \cdot G$. As $S \cdot G$ has rank n_0 (in fact it contains I_{n_0} as a submatrix), the decrypter can obtain σ .

Matrix GSW? We can attempt to use the same GSW invariant (1) to encrypt matrices, where a ciphertext matrix C GSW-encrypts a plaintext matrix M if $S \cdot C = M \cdot S \cdot G + E \pmod q$ for a small noise matrix E . The exact same decryption procedure as above works also in this case, allowing the decrypter to recover E , then $M \cdot S \cdot G$, and then M .

However, the encryption procedure above does not work for matrices in general, it is unclear how to obtain such a GSW-encryption C of M when M is not a scalar matrix (i.e., of the form $\sigma \cdot I$). If we want to set $C = M' \cdot G + P \cdot X$ as before, we need M' to satisfy $S \cdot M' = M \cdot S$, and finding such an M' seems to require knowing S . (For a scalar matrix $M = \sigma \cdot I$, M' is just the scalar matrix with the same scalar, but in a larger dimension.) Hiromasa et al. [20] show how to obtain a version of GSW that encrypts non-scalar matrices, assuming LWE and a circular security assumption. In our context, our GSW ciphertexts only encrypt scalars so we rely just on LWE without circular encryptions.

Homomorphic Operations in GSW. Suppose we have C_1 and C_2 that GSW-encrypt M_1 and M_2 respectively (scalar matrices or otherwise). Then clearly

$C_1 + C_2$ GSW-encrypts $M_1 + M_2$, provided that the sum of errors remains β -bounded. For multiplication, set $C^\times = C_1 \cdot G^{-1}(C_2) \bmod q$. We have:

$$S \cdot C^\times = (M_1 \cdot S \cdot G + E_1) \cdot G^{-1}(C_2) = M_1 \cdot M_2 \cdot S \cdot G + M_1 \cdot E_2 + E_1 \cdot G^{-1}(C_2).$$

Thus, C^\times GSW-encrypts $M_1 \cdot M_2$ provided that the new error $E' = M_1 \cdot E_2 + E_1 \cdot G^{-1}(C_2)$ remains β -bounded. In the new error, the term $E_1 \cdot G^{-1}(C_2)$ is only slightly larger than the original error E_1 , since $G^{-1}(C_2)$ has small coefficients. To keep the term $M_1 \cdot E_2$ small, there are two strategies. First, if M_1 corresponds to a small scalar – e.g., 0 or 1 – then this term is as small as the original error inside C_2 . Second, if $E_2 = 0$, then this term does not even appear. For example, if we want to homomorphically multiply-by-constant $\sigma_2 \in R_q$, we can just set $C_2 = \sigma_2 \cdot G$ (without any $P \cdot X$), and compute C^\times as above. The plaintext inside C_1 will be multiplied by σ_2 , and the new error will not depend on either σ_1 or σ_2 , which therefore can be arbitrary in R_q .

3 Defining Compressible (F)HE

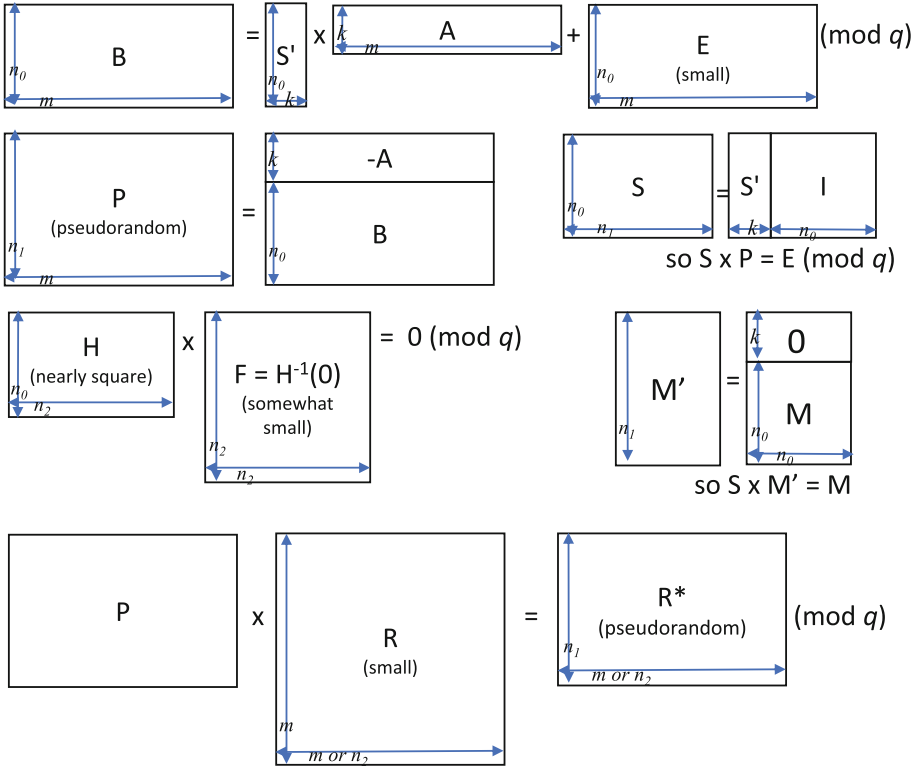
Compressible (F)HE is defined similarly to standard (F)HE, except that decryption is broken into first compression and then “compressed decryption.” Here we present the definition just for the simple case of 1-hop fully homomorphic encryption for bits, but the same type of definition applies equally to multiple hops, different plaintext spaces, and/or partially homomorphic. (See [19] for detailed treatment of all these variations.)

Definition 1. *A compressible fully homomorphic encryption scheme consists of five procedures, (KeyGen, Encrypt, Evaluate, Compress, Decrypt):*

- $(\mathfrak{s}, \text{pk}) \leftarrow \text{KeyGen}(1^\lambda)$. Takes the security parameter λ and outputs a secret/public key-pair.
- $\mathbf{c} \leftarrow \text{Encrypt}(\text{pk}, b)$. Given the public key and a plaintext bit, outputs a low-rate ciphertext.
- $\mathbf{c}' \leftarrow \text{Evaluate}(\text{pk}, \Pi, \mathbf{c})$. Takes a public key pk , a circuit Π , a vector of low-rate ciphertexts $\mathbf{c} = \langle \mathbf{c}_1, \dots, \mathbf{c}_t \rangle$, one for every input bit of Π , and outputs another vector of low-rate ciphertexts \mathbf{c}' , one for every output bit of Π .
- $\mathbf{c}^* \leftarrow \text{Compress}(\text{pk}, \mathbf{c}')$. Takes a public key pk and a vector of low-rate ciphertexts $\mathbf{c}' = \langle \mathbf{c}'_1, \dots, \mathbf{c}'_s \rangle$, and outputs one or more compressed ciphertexts $\mathbf{c}^* = \langle \mathbf{c}^*_1, \dots, \mathbf{c}^*_s \rangle$.
- $\mathbf{b} \leftarrow \text{Decrypt}(\mathfrak{s}, \mathbf{c}^*)$. On secret key and a compressed ciphertext, outputs a string of plaintext bits.

We extend **Decrypt** to a vector of compressed ciphertexts by decrypting each one separately. The scheme is correct if for every circuit Π and plaintext bits $\mathbf{b} = (b_1, \dots, b_t) \in \{0, 1\}^t$, one for every input bit of Π ,

$$\Pr \left[\begin{array}{l} (\mathfrak{s}, \text{pk}) \leftarrow \text{KeyGen}(1^\lambda), \mathbf{c} \leftarrow \text{Encrypt}(\text{pk}, \mathbf{b}), \mathbf{c}' \leftarrow \text{Evaluate}(\text{pk}, \Pi, \mathbf{c}) \\ \quad \quad \quad : \Pi(\mathbf{b}) \text{ is a prefix of } \text{Decrypt}(\mathfrak{s}, \text{Compress}(\text{pk}, \mathbf{c}')) \end{array} \right] = 1. \tag{2}$$



$\sigma \cdot G \in Z_q^{n_1 \times m}$ is a "very redundant" scalar, $C = \sigma G + R^* \pmod{q}$ is a GSW ctxt
 $M^* = M' \times H \in Z_q^{n_1 \times n_2}$ "somewhat redundant" matrix, $C^* = M^* + R^* \pmod{q}$ compressed ctxt

Fig. 1. An illustration of the matrices in our construction. For some small $\epsilon > 0$ we have $n_1 = n_0 + k \approx n_2 = n_0(1 + \epsilon/2)$ and $m = n_1 \log q$. So, $n_0 \approx 2k/\epsilon$. Also, for correct decryption of ciphertexts with error E using gadget matrix H we require $\|E\|_\infty < q^{\epsilon/2}$.

(We allow prefix since the output of Decrypt could be longer than the output length of Π .)

The scheme has rate $\alpha = \alpha(\lambda) \in (0, 1)$ if for every circuit Π with sufficiently long output, plaintext bits $\mathbf{b} = (b_1, \dots, b_t) \in \{0, 1\}^t$, and low rate ciphertexts $\mathbf{c}' \leftarrow \text{Evaluate}(\text{pk}, \Pi, \text{Encrypt}(\text{pk}, \mathbf{b}))$ as in Eq. (2) we have

$$|\text{Compress}(\text{pk}, \mathbf{c}')| \cdot \alpha \leq |\Pi(\mathbf{b})|.$$

(We note that a similar approach can be used also when talking about compression of fresh ciphertexts.)

4 Constructing Compressible (F)HE

On a high level, our compressible scheme combines two cryptosystems: One is a low-rate (uncompressed) FHE scheme, which is a slight variant of GSW, and

the other is a new high-rate (compressed) additively-homomorphic scheme for matrices, somewhat similar to the matrix homomorphic encryption of Hiromasa et al. [20]. What makes our scheme compressible is that these two cryptosystems “play nice,” in the sense that they share the same secret key and we can pack many GSW ciphertexts in a single compressed ciphertext.

The low-rate scheme is the GSW variant from Sect. 2 that uses matrix LWE secrets. The secret key is a matrix of the form $S = [S'|I]$, and the public key is a pseudorandom matrix P satisfying $S \times P = E \pmod{q}$, with q the LWE modulus and E a low norm matrix. This low-rate cryptosystem encrypts small scalars (often just bits $\sigma \in \{0, 1\}$), the ciphertext is a matrix C , and the decryption invariant is $SC = \sigma SG + E \pmod{q}$, with G the gadget matrix and E a low-norm matrix.

For the high-rate scheme we describe two variants, both featuring matrices for keys, plaintexts, and ciphertexts. One variant of the high-rate scheme is the PVW batched encryption scheme [30] (in its matrix notations), and another variant uses a new type of “nearly square” gadget matrix. Both variants have the same asymptotic efficiency, but using the gadget matrix seems to yield better concrete parameters, at least for our PIR application. The PVW-based variant is easier to describe, so we begin with it.

4.1 Compressible HE with PVW-Like Scheme

We now elaborate on the different procedures that comprise our compressible homomorphic encryption scheme.

Key Generation. To generate a secret/public key pair we choose two uniformly random matrices $S' \in R_q^{n_0 \times k}$ and $A \in R_q^{k \times m}$ and a small matrix $E \leftarrow \chi^{n_0 \times m}$, and compute the pseudorandom matrix $B := S' \times A + E \in R_q^{n_0 \times m}$.

The secret key is the matrix $S = [S'|I_{n_0}] \in R_q^{n_0 \times n_1}$ and the public key is $P = [\frac{-A}{B}] \in R_q^{n_1 \times m}$, and we have $S \times P = S' \times (-A) + I \times B = E \pmod{q}$.

Encryption and Evaluation. Encryption and decryption of small scalars and evaluation of circuit on them is done exactly as in the GSW scheme. Namely a scalar $\sigma \in R$ is encrypted by choosing a matrix $X \in R^{m \times m}$ with small entries, then outputting the ciphertext $C := \sigma G + PX \pmod{q}$. These low-rate ciphertexts satisfy the GSW invariant, namely $SC = \sigma SG + E \pmod{q}$ with $E \ll q$. These being GSW ciphertexts, encryption provides semantic security under the decision LWE hardness assumption [17].

Evaluation is the same as in GSW, with addition implemented by just adding the ciphertext matrices modulo q and multiplication implemented as $C^\times := C_1 \times G^{-1}(C_2) \pmod{q}$. Showing that these operations maintain the decryption invariant (as long as the encrypted scalars are small) is done exactly as in GSW.

Compression. The crux of our construction is a compression technique that lets us pack many GSW bit encryptions into a single high-rate PVW ciphertext. Let $p < q$ be the plaintext and ciphertext moduli of PVW and denote $f = \lceil q/p \rceil$. (The ciphertext modulus q is the same one that was used for the GSW encryption.) Also denote $\ell = \lceil \log p \rceil$, and consider $\ell \cdot n_0^2$ GSW ciphertexts,

$C_{u,v,w} \in \mathbb{Z}_q^{n_1 \times m}$, $u, v \in [n_0]$, $w \in [\ell]$, each encrypting a bit $\sigma_{i,j,k} \in \{0, 1\}$. Namely we have $S \times C_{u,v,w} = \sigma_{u,v,w} \cdot SG + E_{u,v,w} \pmod{q}$ for low norm matrices $E_{u,v,w}$.

We want to pack all these ciphertexts into a single compressed PVW ciphertext, namely a matrix $C \in \mathbb{Z}_q^{n_1 \times n_0}$ such that $SC = f \cdot Z + E' \pmod{q}$ where $Z \in \mathbb{Z}_p^{n_0 \times n_0}$ is a plaintext matrix whose bit representation contains all the $\sigma_{u,v,w}$'s (and E' is a noise matrix with entries of magnitude less than $f/2$).

Denote by $T_{u,v}$ the square $n_0 \times n_0$ singleton matrix with 1 in entry (u, v) and 0 elsewhere, namely $T_{u,v} = \mathbf{e}_u \otimes \mathbf{e}_v$ (where $\mathbf{e}_u, \mathbf{e}_v$ are the dimension- n_0 unit vectors with 1 in positions u, v , respectively). Also denote by $T'_{u,v}$ the padded version of $T_{u,v}$ with k zero rows on top, $T'_{u,v} = \begin{bmatrix} 0 \\ \mathbf{e}_u \otimes \mathbf{e}_v \end{bmatrix} \in \mathbb{Z}_q^{n_1 \times n_0}$. We compress the $C_{u,v,w}$'s by computing

$$C^* := \sum_{u,v,w} C_{u,v,w} \times G^{-1}(f \cdot 2^w \cdot T'_{u,v}) \pmod{q}. \tag{3}$$

Since $T'_{u,v}$ are $n_1 \times n_0$ matrices, then $G^{-1}(f \cdot 2^w \cdot T'_{u,v})$ are $m \times n_0$ matrices, and since the $C_{u,v,w}$'s are $n_1 \times m$ matrices then $C^* \in \mathbb{Z}_q^{n_1 \times n_0}$, as needed. Next, for every u, v denote $z_{uv} = \sum_{w=0}^{\ell} 2^w \sigma_{u,v,w} \in [p]$, and we observe that

$$\begin{aligned} S \times C^* &= \sum_{u,v,w} S \times C_{u,v,w} \times G^{-1}(f \cdot 2^w \cdot T'_{u,v}) \\ &= \sum_{u,v,w} (\sigma_{u,v,w} S G + E_{u,v,w}) \times G^{-1}(f \cdot 2^w \cdot T'_{u,v}) \\ &= \sum_{u,v,w} f \cdot 2^w \cdot \sigma_{u,v,w} S T'_{u,v} + \overbrace{\sum_{u,v,w} E_{u,v,w} \times G^{-1}(f \cdot 2^w \cdot T'_{u,v})}^{E'} \\ &= f \cdot \sum_{u,v} z_{u,v} S T'_{u,v} + E' \stackrel{(*)}{=} f \cdot \underbrace{\sum_{u,v} z_{u,v} T_{u,v}}_Z + E', \end{aligned} \tag{4}$$

where $Z = [z_{u,v}] \in [p]^{n_0 \times n_0}$. (The equality $(*)$ holds since $S = [S'|I]$ and $T' = \begin{bmatrix} 0 \\ T \end{bmatrix}$ and therefore $ST' = S' \times 0 + I \times T = T$.)

Compressed Decryption. Compressed ciphertexts are just regular PVW ciphertexts, hence we use the PVW decryption procedure. Given the compressed ciphertext $C^* \in \mathbb{Z}_q^{n_1 \times n_0}$, we compute $X := SC = f \cdot Z + E' \pmod{q}$ using the secret key S . As long as $\|E'\|_\infty < f/2$, we can complete decryption by rounding to the nearest multiple of f , setting $Z := \lceil X/f \rceil$. Once we have the matrix Z , we can read off the $\sigma_{u,v,w}$'s which are the bits in the binary expansion of the $z_{u,v}$'s.

Lemma 1. *The scheme above is a compressible (F)HE cryptosystem with rate $\alpha = \frac{|p|}{|q|} \cdot \frac{n_0}{n_1}$. □*

Setting the Parameters. It remains to show how to set the various parameters – including the matrix dimensions n_0, n_1 and the moduli p, q – as a function of the security parameter k . If we use a somewhat-homomorphic variant of GSW

without bootstrapping, then the noise magnitude in evaluated ciphertexts would depend on the functions that we want to compute. One such concrete example (with fully specified constants) is provided in Sect. 5 for our PIR application. Here we provide an asymptotic analysis of the parameters when using GSW as a fully-homomorphic scheme with bootstrapping. Namely we would like to evaluate an arbitrary function with long output on encrypted data (using the GSW FHE scheme), then pack the resulting encrypted bits in compressed ciphertexts that remain decryptable.

We want to ensure that compressed ciphertexts have rate of $1 - \epsilon$ for some small ϵ of our choosing. To this end, it is sufficient to set $n_0 > 2k/\epsilon$ and $q = p^{1+\epsilon/2}$. This gives $n_1 = n_0 + k \leq n_0(1 + \epsilon/2)$ and $|q| = |p|(1 + \epsilon/2)$, and hence

$$\frac{n_0}{n_1} \cdot \frac{|p|}{|q|} \geq \left(\frac{1}{1 + \epsilon/2}\right)^2 > (1 - \epsilon/2)^2 > 1 - \epsilon,$$

as needed.

Using $q = p^{1-\epsilon/2}$ means that to be able to decrypt we must keep the noise below $q/2p = p^{\epsilon/2}/2$. Following [11, 17], when using GSW with fresh-ciphertext noise of size α and ciphertext matrices of dimension $n_1 \times m$, we can perform arbitrary computation and then bootstrap the result, and the noise after bootstrapping is bounded below αm^2 . From Eq. (4) we have a set of $n_0^2 \log p$ error matrices $E_{u,v,w}$, all satisfying $\|E_{u,v,w}\|_\infty < \alpha m^2$. The error term after compression is therefore $\sum_{u,v,w} E_{u,v,w} G^{-1}(\text{something})$, and its size is bounded by $n_0^2 \log p \cdot \alpha m^2 \cdot m = \alpha m^3 n_0^2 \log p$.

It is enough, therefore, that this last expression is smaller than $p^{\epsilon/2}/2$, i.e., we have the correctness constraint $p^{\epsilon/2}/2 > \alpha m^3 n_0^2 \log p$. Setting the fresh-encryption noise as some polynomial in the security parameter, the last constraint becomes $p^{\epsilon/2} > \text{poly}(k) \log p$. This is satisfied by some $p = k^{\Theta(1/\epsilon)}$, and therefore also $q = p^{1+\epsilon/2} = k^{\Theta(1/\epsilon)}$.

We conclude that to get a correct scheme with rate $1 - \epsilon$, we can use LWE with noise $\text{poly}(k)$ and modulus $q = k^{\Theta(1/\epsilon)}$. Hence the security of the scheme relies on the hardness of LWE with gap $k^{\Theta(1/\epsilon)}$, and in particular if ϵ is a constant then we rely on LWE with polynomial gap.

We note that there are many techniques that can be applied to slow the growth of the noise. Many of those techniques (for example modulus switching) are described in Sect. 5 in the context of our PIR application. While they do not change the asymptotic behavior — we will always need $q = k^{\Theta(1/\epsilon)}$ — they can drastically improve the constant in the exponent.

Theorem 1. *For any $\epsilon = \epsilon(\lambda) > 0$, there exists a rate- $(1 - \epsilon)$ compressible FHE scheme as per definition 1 with semantic security under the decision-LWE assumption with gap $\text{poly}(\lambda)^{1/\epsilon}$. \square*

More Procedures. In addition to the basic compressible HE interfaces, our scheme also supports several other operations that come in handy in applications such as PIR.

Encryption and additive homomorphism of compressed ciphertexts.

Since this variant uses PVW for compressed ciphertexts, then we can use the encryption and additive homomorphism of the PVW cryptosystem.

Multiplying compressed ciphertexts by GSW ciphertexts. When p divides q , we can also multiply a compressed ciphertext $C' \in \mathbb{Z}_q^{n_1 \times n_0}$ encrypting $M \in \mathbb{Z}_p^{n_0 \times n_0}$ by a GSW ciphertext $C \in \mathbb{Z}_q^{n_1 \times m}$ encrypting a small scalar σ , to get a compressed ciphertext C'' that encrypts the matrix $\sigma M \bmod p$. This is done by setting $C'' := C \times G^{-1}(C') \bmod q$ (and note that $C' \in \mathbb{Z}_q^{n_1 \times n_0}$ so $G^{-1}(C') \in \mathbb{Z}_q^{m \times n_0}$). For correctness, recall that we have $SC = \sigma SG + E$ and $SC' = q/p \cdot M + E'$ over \mathbb{Z}_q , hence

$$\begin{aligned} S \times C'' &= S C G^{-1}(C') = \sigma SC' + \overbrace{E G^{-1}(C')}^{E''} \\ &= \sigma(q/p \cdot M + E') + E'' = q/p \cdot (\sigma M \bmod p) + \overbrace{\sigma E' + E''}^{E^*} \pmod{q}. \end{aligned} \quad (5)$$

This is a valid compressed encryption of $\sigma M \bmod p$ as long as the noise $E^* = \sigma E' + E G^{-1}(C')$ is still smaller than $p/2$.

Multiplying GSW ciphertexts by plaintext matrices. The same technique that lets us right-multiply GSW ciphertexts by compressed ones, also lets us right-multiply them by plaintext matrices. Indeed if $M \in \mathbb{Z}_p^{n_0 \times n_0}$ is a plaintext matrix and M' is its padded version $M' = \begin{bmatrix} 0 \\ M \end{bmatrix} \in \mathbb{Z}_p^{n_1 \times n_0}$, then the somewhat redundant matrix $M^* = q/p \cdot M'$ can be considered a noiseless ciphertext (note that $S \times M^* = q/p \cdot M$) and can therefore be multiplied by a GSW ciphertext as above. The only difference is that in this case we can even use a GSW ciphertext encrypting a large scalar: The “noiseless ciphertext” M^* has $E' = 0$, hence the term $\sigma E'$ from above does not appear in the resulting noise term, no matter how large σ is.

4.2 High-Rate Additive HE Using Nearly Square Gadget Matrix

We now turn to the other variant of our scheme. Here we encrypt plaintext matrices modulo q using ciphertext matrix modulo the same q , with dimensions that are only slightly larger than the plaintext matrix. A new technical ingredient in that scheme is a new gadget matrix (described in Sect. 4.4), that we call H : Just like the G gadget matrix from [27], our H adds redundancy to the ciphertext, and it has a “public trapdoor” that enables removing the noise upon decryption. The difference is that H is a *nearly square matrix*, hence comes with almost no expansion, enabling high-rate ciphertexts. Of course, an almost rectangular H cannot have a trapdoor of high quality, so we make do with a low-quality trapdoor that can only remove a small amount of noise.

The slight increase in dimensions from plaintext to ciphertext in this high-rate scheme comes in two steps. First, as in the previous variant we must pad plaintext matrices M with some additional zero rows, setting $M' = \begin{bmatrix} 0 \\ M \end{bmatrix}$ so as to get $SM' = M$. Second, we add redundancy to M' by multiplying it on the right

by our gadget matrix H , to enable removing a small amount of noise during decryption. The decryption invariant for compressed ciphertexts is

$$S \times C = M \times H + E \pmod{q},$$

with $S = (S'|I)$ the secret key, C the ciphertext, M the plaintext matrix and E a small-norm noise matrix.

To get a high-rate compressed ciphertexts, we must ensure that the increase in dimensions from plaintext to ciphertext is as small as possible. With $n_0 \times n_0$ plaintext matrices M , we need to add as many zero rows as the dimension of the LWE secret (which we denote by k). Denoting $n_1 = n_0 + k$, the padded matrix M' has dimension $n_1 \times n_0$. We further add redundancy by multiplying on the right with a somewhat rectangular gadget matrix H of dimension $n_0 \times n_2$. The final dimension of the ciphertext is $n_1 \times n_2$, so the information rate of compressed ciphertexts is $n_0^2 / (n_1 n_2)$. As we show in Sect. 4.3, we can orchestrate the various parameters so that we can get $n_0^2 / (n_1 n_2) = 1 - \epsilon$ for any desired $\epsilon > 0$, using a modulus q of size $k^{\Theta(1/\epsilon)}$. Hence we can get any constant $\epsilon > 0$ assuming the hardness of LWE with polynomial gap, or polynomially small ϵ if we assume hardness of LWE with subexponential gap.

The rest of this section is organized as follows: We now describe on the different procedures that comprise this variant, then discuss parameters and additional procedures, and finally in Sect. 4.4 we describe the construction of the gadget matrix H .

Key Generation, Encryption, and Evaluation. These are identical to the procedures in the variant from Sect. 4.1, using GSW with matrix secret keys. The low-rate ciphertexts satisfy the GSW invariant as GSW, $SC = \sigma SG + E \pmod{q}$ with $E \ll q$, and provides semantic security under the decision LWE hardness assumption [17].

Compression. Compression is similar to the previous variant, but instead of $G^{-1}(f \cdot 2^w \cdot T'_{u,v})$ as in Eq. (3) we use $G^{-1}(2^w \cdot T'_{u,v} \times H)$. Recall that we denote by $T_{u,v}$ the square $n_0 \times n_0$ singleton matrix with 1 in entry (u, v) and 0 elsewhere, and $T'_{u,v}$ is a padded version of $T_{u,v}$ with k zero rows on top

Denote $\ell = \lfloor \log q \rfloor$, and consider $\ell \cdot n_0^2$ GSW ciphertexts, $C_{u,v,w} \in \mathbb{Z}_q^{n_1 \times m}$, $u, v \in [n_0], w \in [\ell]$, each encrypting a bit $\sigma_{i,j,k} \in \{0, 1\}$, we pack these GSW bit encryptions into a single compressed ciphertext by computing

$$C^* = \sum_{u,v,w} C_{u,v,w} \times G^{-1}(2^w \cdot T'_{u,v} \times H) \pmod{q},$$

We first note that $T'_{u,v} \times H$ are $n_1 \times n_2$ matrices, hence $G^{-1}(2^w \cdot T'_{u,v} \times H)$ are $m \times n_2$ matrices, and since the $C_{u,v,w}$'s are $n_1 \times m$ matrices then $C^* \in \mathbb{Z}_q^{n_1 \times n_2}$, as needed. Next, for every u, v denote $z_{uv} = \sum_{w=0}^{\ell} 2^w \sigma_{u,v,w} \in [q]$, and

we observe that

$$\begin{aligned}
 S \times C^* &= \sum_{u,v,w} S \times C_{u,v,w} \times G^{-1}(2^w \cdot T'_{u,v} \times H) \\
 &= \sum_{u,v,w} (\sigma_{u,v,w} S G + E_{u,v,w}) \times G^{-1}(2^w \cdot T'_{u,v} \times H) \\
 &= \sum_{u,v,w} 2^w \sigma_{u,v,w} S T'_{u,v} H + \overbrace{\sum_{u,v,w} E_{u,v,w} \times G^{-1}(2^w \cdot T'_{u,v} \times H)}^{E'} \\
 &= \sum_{u,v} z_{u,v} S T'_{u,v} H + E' \stackrel{(*)}{=} \left(\overbrace{\sum_{u,v} z_{u,v} T'_{u,v}}^Z \right) \times H + E', \tag{6}
 \end{aligned}$$

where $Z = [z_{u,v}] \in [q]^{n_0 \times n_0}$. (The equality $(*)$ holds since $S = [S'|I]$ and $T' = \begin{bmatrix} 0 \\ T \end{bmatrix}$ and therefore $ST' = S' \times 0 + I \times T = T$.)

Compressed Decryption. Compressed ciphertexts in this scheme are matrices $C \in \mathbb{Z}_q^{n_1 \times n_2}$, encrypting plaintext matrices $M \in \mathbb{Z}_q^{n_0 \times n_0}$. To decrypt we compute $X := \hat{S} C = M H + E \pmod q$ using the secret key S . This is where we use the redundancy introduced by H , as long as $\|E\|_\infty$ is small enough, we can complete decryption by using the trapdoor $F = H^{-1}(0)$ to recover and then eliminate the small noise E , hence obtaining the matrix M . This recovers the matrix Z , and then we can read off the $\sigma_{u,v,w}$'s which are the bits in the binary expansion of the $z_{u,v}$'s.

Lemma 2. *The scheme above is a compressible FHE scheme with rate $\alpha = n_0^2/n_1 n_2$.* □

More Procedures. It is easy to see that the current construction supports the same additional procedures as the variant from Sect. 4.1. Namely we have direct encryption and additive homomorphism of compressed ciphertexts, multiplication of compressed ciphertexts by GSW ciphertexts that encrypts small constants, and multiplication of GSW ciphertexts (encrypting arbitrary constants) by plaintext mod- q matrices.

4.3 Setting the Parameters

It remains to show how to set the various parameters — the dimensions n_0, n_1, n_2 and the modulus q — as a function of the security parameter k . As above, we only provide here an asymptotic analysis of the parameters when using GSW as a fully-homomorphic scheme with bootstrapping.

Again from [11, 17], if we use fresh-ciphertext noise of size $\text{poly}(k)$ then also after bootstrapping we still have the noise magnitude bounded below $\text{poly}(k)$. After compression as per Eq. (6), the noise term is a sum of $n_0^2 \log q$ matrices $E_{u,v,w}$, all of magnitude bounded by $\text{poly}(k)$, hence it has magnitude below $\text{poly}(k) \cdot \log q$. We therefore need the nearly-square gadget matrix H to add enough redundancy to correct that noise.

On the other hand, to get an information rate of $1 - \epsilon$ (for some small ϵ) we need $n_0^2/(n_1 n_2) \geq 1 - \epsilon$, which we can get by setting $n_1, n_2 \leq n_0/(1 - \frac{\epsilon}{2})$. As we explain in Sect. 4.4 below, a nearly-square matrix H with $n_2 = n_0/(1 - \frac{\epsilon}{2})$ can only correct noise of magnitude below $\beta = \lfloor q^{\epsilon/2}/2 \rfloor$. Hence we get the correctness constraint $\frac{q^{\epsilon/2}}{2} > \text{poly}(k) \log q$ (essentially the same as for the variant from Sect. 4.1 above), which is satisfied by some $q = k^{\Theta(1/\epsilon)}$.

4.4 A Nearly Square Gadget Matrix

We now turn to describing the new technical component that we use in the second variant above, namely the “nearly square” gadget matrix. Consider first why the usual Micciancio-Peikert gadget matrix [27] $G \in \mathbb{Z}_q^{n_1 \times m}$ which is used GSW cannot give us high rate. An encryption of $M \in R_q^{n_0 \times n_0}$ has the form $C = M' \cdot G + P \cdot X$ (for some M' that includes M), so the rate can be at most n_0/m simply because C has m/n_0 times as many columns as M . This rate is less than $1/\log q$ for the usual G .

The rate can be improved by using a “lower-quality” gadget matrix. For example $G = I \otimes \mathbf{g}$ where $\mathbf{g} = (1, B, \dots, B^{\lfloor \log_B q \rfloor})$ for large-ish B , where $G^{-1}(C)$ still have coefficients of magnitude at most $B/2$. But this can at best yield a rate- $1/2$ scheme (for $B = \sqrt{q}$), simply because a non-trivial \mathbf{g} must have dimension at least 2. Achieve rate close to 1 requires that we use a gadget matrix with almost the same numbers of rows and columns.

The crucial property of the gadget matrix that enables decryption, is that there exists a known “public trapdoor” matrix $F = G^{-1}(0) \in R^{m \times m}$ such that:

1. F has small entries ($\ll q$)
2. $G \cdot F = 0 \pmod q$
3. F is full-rank over R (but of course not over R_q , as it is the kernel of G).

Given such an F , it is known how to compute $G^{-1}(C)$ for any ciphertext $C \in R_q^{n_1 \times m}$, such that the entries in $G^{-1}(C)$ are not much larger than the coefficients of F , cf. [16].

In our setting, we want our new gadget matrix (that we call H rather than G to avoid confusion) to have almost full rank modulo q (so that it is “nearly square”), hence we want $F = H^{-1}(0)$ to have very low rank modulo q . Once we have a low-norm matrix F with full rank over R but very low rank modulo q , we simply set H as a basis of the mod- q kernel of F .

Suppose for simplicity that $q = p^t - 1$ for some integers p, t . We can generate a matrix F' with “somewhat small” coefficients that has full rank over the reals but rank one modulo q as:

$$F' := \begin{bmatrix} 1 & p & p^2 & \dots & p^{t-1} \\ p^{t-1} & 1 & p & \dots & p^{t-2} \\ p^{t-2} & p^{t-1} & 1 & \dots & p^{t-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p & p^2 & p^3 & \dots & 1 \end{bmatrix}$$

Notice that the entries of F' have size at most $(q+1)/p \approx q^{1-1/t}$ and moreover for every vector \mathbf{v} we have

$$\|\mathbf{v}F'\|_\infty \leq \|\mathbf{v}\|_\infty \cdot (1+p+\dots+p^{t-1}) = \|\mathbf{v}\|_\infty \cdot (p^t-1)/(p-1) = \|\mathbf{v}\|_\infty \cdot \frac{q}{p-1}. \quad (7)$$

We can use this F' to generate a matrix F with rank $r \cdot t$ over the reals but rank r modulo q (for any r), by tensoring F' with the $r \times r$ identity matrix, $F := F' \otimes I_r$. This yields the exact same bounds as above on the l_∞ norms. Our gadget matrix H is an $r(t-1) \times rt$ matrix whose rows span the null space of F modulo q (any such matrix will do). For our scheme below we will set $n_0 = r(t-1)$ and $n_2 = rt = n_0(1 + \frac{1}{t-1})$.

In the decryption of compressed ciphertexts below, we use the ‘‘somewhat smallness’’ of $F = H^{-1}(0)$. Specifically, given a matrix $Z = MH + E \pmod{q}$ with $\|E\|_\infty \leq \frac{p-1}{2}$, we first multiply it by F modulo q to get $ZF = (MH+E)F = EF \pmod{q}$ (since $HF = 0 \pmod{q}$). But

$$\|EF\|_\infty \leq \|E\|_\infty \cdot \frac{q}{p-1} \leq \frac{p-1}{2} \cdot \frac{q}{p-1} = q/2,$$

and therefore $(ZF \pmod{q}) = EF$ over the integers. Now we use the fact that F has full rank over the reals, and recover $E := (ZF \pmod{q}) \times F^{-1}$. Then we compute $Z - E = MH \pmod{q}$, and since H has rank n_0 modulo q we can recover M from MH . It follows that to ensure correctness when decrypting compressed ciphertexts, it is sufficient to use a bound $\beta \leq \frac{p-1}{2} = \lfloor q^{1/t} \rfloor / 2$ on the size of the noise in compressed ciphertexts.

The restriction $q = p^t - 1$ is not really necessary; many variants are possible. The following rather crude approach works for any q that we are likely to encounter. Consider the lattice L of multiples of the vector $\mathbf{u} = (1, a, \dots, a^{t-1})$ modulo q , where $a = \lceil q^{1/t} \rceil$. Let the rows of F' be the L -vectors $c_i \cdot \mathbf{u} \pmod{q}$ for $i \in [t]$, where $c_i = \lceil q/a^i \rceil$. Clearly F' has rank 1 modulo q . (We omit a proof that F' is full rank over the integers.) We claim that all entries of F' are small. Consider the j -th coefficient of $c_i \cdot \mathbf{u} \pmod{q}$, which is $\lceil q/a^i \rceil \cdot a^j \pmod{q}$ for $i \in [t]$, $j \in \{0, \dots, t-1\}$. If $i > j$, then $\lceil q/a^i \rceil \cdot a^j$ is bounded in magnitude by $q/a^{i-j} + a^j \leq q/a + a^{t-1} \leq 2a^{t-1}$. For the $j \geq i$ case, observe that $\lceil q/a^i \rceil \cdot a^i$ is an integer in $[q, q + a^i]$, and therefore is at most a^i modulo q . Therefore $\lceil q/a^i \rceil \cdot a^j \pmod{q}$ is at most $a^j \leq a^{t-1}$ modulo q . As long as $q \geq t^t$, we have that $a^{t-1} \leq (q^{1/t} \cdot (1+1/t))^{t-1} < q^{(t-1)/t} \cdot e$ – that is, $\|F'\|_\infty$ is nearly as small as it was when we used $q = p^t - 1$. As we saw above, q anyway needs to exceed β^t where β is a bound on the noise of ciphertexts, so the condition that $q > t^t$ will likely already be met.

5 Application to Fast Private Information Retrieval

Can we construct a single-server PIR scheme that is essentially optimal both in terms of communication and computation? With our compressible FHE scheme, we can achieve communication rate arbitrarily close to 1. Here, we describe a PIR

which is not only bandwidth efficient but should also outperform whole-database AES encryption computationally.⁶

5.1 Toward an Optimized PIR Scheme

Our starting point is the basic hierarchical PIR, where the N database entries are arranged in a hypercube of dimensions $N = N_1 \times \dots \times N_D$ and the scheme uses degree- D homomorphism:

- The client’s index $i \in [N]$ is represented in mixed radix of basis N_1, \dots, N_D , namely as (i_1, \dots, i_D) such that $i = \sum_{j=1}^D i_j \cdot \prod_{k=j+1}^D N_k$. The client’s message is processed to obtain an encrypted *unary representation* of all the i_j ’s. Namely, for each dimension j we get a dimension- N_j vector of encrypted bits, in which the i_j ’th bit is one and all the others are zero.
- Processing the first dimension, we multiply each hyper-row $u \in [N_1]$ by the u ’th encrypted bit from the first vector, which zeros out all but the i_1 ’st hyper-row. We then add all the resulting encrypted hyper-rows, thus getting a smaller hypercube of dimension $N/N_1 = N_2 \times \dots \times N_D$, consisting only the i_1 ’st hyper-row of the database.
- We proceed in a similar manner to fold the other dimensions, one at a time, until we are left with a zero-dimension hypercube consisting only the selected entry i .

We note that the first step, reducing database size from N to N/N_1 , is typically the most expensive since it processes the most data. On the other hand, that step only requires ciphertext-by-plaintext multiplications (vs. the ciphertext-by-ciphertext multiplications that are needed in the following steps), so it can sometimes be optimized better than the other steps.

Below we describe the sequence of derivations and optimizations to get our final construction, resulting in a high rate PIR scheme which is also computationally efficient. The construction features a tradeoff between bandwidth and computation (and below we describe a variant with rate $4/9$).

The main reason for this tradeoff is that the rate of our scheme is $\frac{n_0}{n_1} \cdot \frac{n_0}{n_2}$, where the secret key matrix S has dimension $n_0 \times n_1$ and the gadget matrix H has dimension $n_0 \times n_2$. Since n_0, n_1, n_2 are integers, we need n_0 to be large if we want n_0/n_1 and n_0/n_2 to be close to one. Recalling that the plaintext matrices M have dimension $n_0 \times n_0$, a large n_0 means that the plaintext is of high dimension. Hence multiplying GSW-ciphertexts C by plaintext matrices M takes more multiplications per entry (e.g., using a cubic matrix multiplication algorithm). A second aggravating factor is that as H becomes closer to square, we can handle smaller noise/modulus ratio. Hence we need the products $C \times M$

⁶ The “should” is since we did not implement this construction. Implementing it and measuring its performance may be an interesting topic for future work.

to be carried over a larger modulus (so we can later mod-switch it down to reduce the noise), again getting more multiplies per plaintext byte.⁷

Using Our GSW-Compatible Compressible HE Scheme. An advantage of GSW over other FHE schemes is its exceptionally slow noise growth during homomorphic multiplication when the left multiplicand is in $\{0, 1\}$. Although GSW normally operates on encrypted bits, GSW’s advantage remains when the right multiplicand is a ciphertext of our compressible FHE scheme. So, these schemes are perfect for PIR, where the left multiplicands are bits of the client’s query, and the rightmost multiplicands are blocks of the database.

Using Ring-LWE. As usual with LWE schemes, we can improve performance by switching to the ring (or module) variant, where the LWE secret has low dimension over a large extension field. Instead of having to manipulate large matrices, these variants manipulate low-dimension matrices over the same large extension field, which take less bits to describe and can be multiplied faster (using FFTs). To get comparable security, if the basic LWE scheme needs LWE secrets of dimension k , the new scheme will have dimension- k' secrets over an extension field of degree d , such that $k'd \geq k$. (For ring-LWE we have $k' = 1$ and $d = k$.) The various matrices in the scheme consist of extension-field elements, and their dimensions are $n'_i = n_i/d$ and $m' = m/d$ (instead of n_i, m , respectively). Below we use the notation n'_i and m' to emphasize the smaller values in the RLWE context.

Saving on FFTs. One of our most important optimizations is pre-processing the database to minimize the number of FFTs during processing. Our scheme needs to switch between CRT representation of ring elements (which is needed for arithmetic operations) and representation in the decoding basis (as needed for applications of $G^{-1}(\cdot)$). While converting between the two can be done in quasi-linear time using FFTs, it is still by far the most expensive operations used in the implementation. (For our typical sizes, converting an element between these representations is perhaps 10–20 times slower than multiplying two elements represented in the CRT basis.)

As in the XPIR work [1], we can drastically reduce the number of FFTs by pre-processing the database, putting it all in CRT representation. This way, we only need to compute FFTs when we process the client’s message to get the encrypted unary representation of the i_j ’s (which is independent of the size of entries in the database), and then again after we fold the first dimension (so it is only applied to compressed ciphertexts encrypting the N/N_1 database entries).

If we set N_1 large enough relative to the FFT overhead, then the FFTs after folding the first dimension will be amortized and become insignificant. On the other hand we need to set it small enough (relative to N/N_1 and the length- L of the entries) so the initial FFTs (of which we have about $n'_1 \cdot m' \cdot N_1$) will also be insignificant.

⁷ The tradeoffs become harder to describe cleanly when optimizing concrete performance as we do here. For example, a 65-bit modular multiplication is as expensive in software as a 120-bit one.

In the description below we illustrate the various parameters with $N_1 = 2^8$, which seems to offer a good tradeoff. For the other N_i 's, there is (almost) no reason to make them large, so we use $N_2 = N_3 = \dots = N_D = 4$. We note that for the construction below there is (almost) no limit on how many such small N_i 's we can use. Below we illustrate the construction for a database with $N = 2^{20}$ entries, but it can handle *much* larger databases. (The same parameters work upto at least $N = 2^{20}$ entries.)

Client-side Encryption. In the context of a PIR scheme, the encrypter is the client who has the decryption key. Hence it can create ciphertexts using the secret key, by choosing a fresh pseudorandom public key P_i for each ciphertext and setting $C_i := \sigma_i G + P_i \bmod q$. This results in ciphertexts of slightly smaller noise, namely just the low-norm E_i 's (as opposed to $E \times X_i$ that we get from public-key encrypted ciphertexts).

Since our PIR construction uses small dimensions $N_2 = N_3 = \dots = 4$, we have the client directly sending the encrypted unary vectors for these dimensions. Namely for each $j = 2, 3, \dots$ the client sends four ciphertexts $C_{j,0}, \dots, C_{j,3}$ such that C_{j,i_j} encrypts one and the others encrypt zero.

For the first dimension we have a large $N_1 = 2^8$, however, so the client sends encryptions of the bits of i_1 and we use the GSW homomorphism to compute the encrypted unary vector for this dimension. Overall the client therefore sends $\log N_1 + (N_2 + N_3 + \dots + N_D)$ encrypted bits, in our illustrated sizes this comes up to $8 + 4 \times 6 = 32$ encrypted bits.

Multiple G Matrices. The accumulated noise in our scheme has many terms of the form $E \times G^{-1}(\text{something})$, but not all of them are created equal. In particular, when folding the first (large) dimension N_1 , the GSW ciphertexts are evaluated and the noise in them is itself a sum of such. When we multiply these GSW ciphertexts by the plaintext matrix we get $E \times G^{-1}(\text{something}) \times G^{-1}(\text{something}')$, which is larger. For the other (small) dimensions, on the other hand, we multiply by fresh ciphertexts so we get much smaller noise. This imbalance leads to wasted resources.

Moreover, the multiplication by $G^{-1}(\text{something})$ during the initial processing of the client's bits are only applied to a small amounts of data. But the multiplication between the GSW matrices and the plaintext data touches all the data in the database. Hence the latter are much more expensive, and we would like to reduce the dimension of the matrices involved as much as we can.

For all of these reasons, it is better to use different G matrices in different parts of the computation. In particular we use very wide-and-short G matrices (with smaller norm of $G^{-1}(0)$) when we initially process the client's bits, and more-square/higher-norm G matrices later on.

Modulus Switching. Even with a careful balance of the G matrices, we cannot make the noise as small as we want it to be for our compressed scheme. We therefore use the modulus-switching technique from [9, 10]. Namely we perform the computation relative to a large modulus Q , then switch to a smaller modulus q before sending the final result to the client, scaling the noise roughly by q/Q .

This lets us be more tolerant to noise, which improves many of the parameters. For example by using $Q \approx q^{2.5}$ we can even replace the G matrix for the actual data by the identity matrix. Even if it means using LWE secret of twice the dimension and having to write numbers that are more than twice as large, it would still save a large constant factor. Moreover it lets us use a more square matrix H (e.g. 2×3) thereby getting a higher rate.

We note that using modulus switching requires that we choose the secret key from the error distribution rather than uniformly. (Also, in the way we implement it, for some of the bits σ we encrypt the scalar $q' \cdot \sigma$ rather than σ itself, where $Q = q' \cdot q$.)

5.2 The Detailed PIR Scheme

Our construction is staged in the cyclotomic ring of index 2^{13} and dimension 2^{12} , i.e., $R = \mathbb{Z}[X]/(X^{2^{12}} + 1)$. The ciphertext modulus of the fresh GSW ciphertext is a composite $Q = q \cdot q'$, with $q \approx 2^{46}$ and $q' \approx 2^{60}$ (both with primitive 2^{12} 'th roots of unity so it is easy to perform FFTs modulo q, q'). Below we denote the rings modulo these three moduli by $R_Q, R_q, R_{q'}$.

We use ring-LWE over R_Q , in particular our LWE secret is a scalar in R_Q , chosen from the error distribution [4]. (Consulting Table 1 from [2], using this cyclotomic ring with a modulus Q of size up to 111 bits yields security level of 128 bits.)

For the various matrices in our construction we use dimensions $k' = 1, n'_0 = 2$, and $n'_1 = n'_2 = 3$, and the plaintext elements are taken from R_q . Hence we get a rate of $(\frac{2}{3})^2 \approx 0.44$. While processing, however, most ciphertexts will be modulo the larger $Q = q \cdot q'$, it is only before we send to the clients that we mod-switch them down to q . We use the construction from Sect. 4.4 with a 2-by-3 matrix H .

We split a size- N database into a hypercube of dimensions $N = 256 \times 4 \times 4 \times \dots \times 4$. A client wishing to retrieve an entry $i \in [N]$ first represents i as (i_1, i_2, \dots, i_D) , with $i_i \in [256]$ and $i_j \in [4]$ for all $j > 1$. Let $\sigma_{1,0}, \dots, \sigma_{1,7}$ be the bits of i_1 , the client then encrypts the scalars $q' \cdot \sigma_{1,0}$ and $\sigma_{1,1}, \dots, \sigma_{1,7}$ in GSW ciphertexts (modulo Q). For $j = 2, \dots, D$ the client uses GSW ciphertexts to encrypt the bits of the unit vector e_{i_j} which is 1 in position i_j and zero elsewhere. We use three different gadget matrices for these GSW ciphertexts:

- For the LSB of i_1 (which will be the rightmost bit to be multiplied using GSW) we eliminate that gadget matrix G altogether and just use the identity, but we also multiply the bit $\sigma_{1,0}$ by q' . Namely we have $C_{1,0} \in R_Q^{n'_0 \times n'_1}$ such that $SC_{1,0} = \sigma_{1,0} q' S + E \in R_Q^{n'_0 \times n'_1}$.
- For the other bits of i_1 we use a wide and short $G_1 \in \mathbb{Z}^{n'_1 \times m'_1}$, where $m'_1 = n'_1 \lceil \log_4 Q \rceil = 3 \cdot 53 = 159$. Each bit $\sigma_{1,t}$ is encrypted by $C_{1,t} \in R_Q^{n'_1 \times m'_1}$ such that $SC_{1,t} = \sigma_{1,t} S G_1 + E \pmod{Q}$.
- For the bits encoding the unary representation of the other i_j 's ($j > 1$), we use a somewhat rectangular (3-by-6) matrix $G_2 \in \mathbb{Z}^{n'_1 \times m'_2}$, where $m'_2 = n'_1 \lceil \log_{2^{53}}(Q) \rceil = 3 \cdot 2 = 6$.

The client sends all these ciphertexts to the server. The encryption of the bits of i_1 consists of 9 elements for encrypting the LSB and $7 \cdot 3 \cdot 159 = 3381$ elements for encrypting the other seven bits. For each of the other indexes i_j we use $4 \cdot 3 \cdot 6 = 72$ elements to encrypt the unary representation of i_j . In our numerical example with $N = 2^{20}$ database entries we have 6 more i_j 's, so the number of ring elements that the client sends is $9 + 3381 + 6 \cdot 72 = 3822$. Each element takes $106 \cdot 2^{12}$ bits to specify, hence the total number of bits sent by the client is $106 \cdot 2^{12} \cdot 3822 \approx 2^{30.6}$ (a bulky 198 MB).

For applications where the client query size is a concern, we can tweak the parameter, e.g. giving up a factor of 2 in the rate, and getting a 2–4× improvement in the client query size. A future-work direction is to try and port the query-expansion technique in the SealPIR work [3] in our setting, if applicable it would yield a very significant reduction in the client query size.⁸

The server pre-processes its database by breaking each entry into 2-by-2 plaintext matrices over R_q (recall $q \approx 2^{46}$). Hence each matrix holds $2 \cdot 2 \cdot 46 \cdot 2^{12} \approx 2^{19.5}$ bits (92KB). The server encodes each entry in these matrices in CRT representation modulo Q .⁹ Below we let L be the number of matrices that it takes to encode a single database entry. (A single JPEG picture will have $L \approx 4$, while a 4 GB movie will be encoded in about 44 K matrices).

Given the client's ciphertext, the server uses GSW evaluation to compute the GSW encryption of the unit vector e_{i_1} for the first dimension (this can be done using less than $N_1 = 256$ GSW multiplications). For $r = 1, 2, \dots, 256$ the server multiplies the r 'th ciphertext in this vector by all the plaintext matrices of all the entries in the r 'th hyperrow of the hypercube, and adds everything across the first hypercube dimension. The result is a single encrypted hyperrow (of dimensions $N_2 \times \dots \times N_D$), each entry of which consists of L compressed ciphertexts.

The server next continues to fold the small dimensions one after the other. For each size-4 dimension it multiplies the four GSW-encrypted bits by all the compressed ciphertexts in the four hyper-columns, respectively, then adds the results across the current dimension, resulting in a 4-fold reduction in the number of ciphertexts. This continues until the server is left with just a single entry of L compressed ciphertexts modulo Q .

Finally the server performs modulus switching, replacing each ciphertext C by $C' = \lceil C/q' \rceil \in R_{q'}$, and sends the resulting ciphertexts to the client for decryption. Note that the ciphertext C satisfied $SC = q'MH + E \pmod{q'q}$. Denoting the rounding error by Ξ , the new ciphertext has

$$SC' = S(C/q' + \Xi) = MH + E/q' + S\Xi \pmod{q'}.$$

⁸ Using the SealPIR optimization requires a key-switching mechanism for GSW, which is not straightforward.

⁹ While the entries in the plaintext matrices are small (in $[\pm 2^{45}]$), their CRT representation modulo Q is not. Hence this representation entails a $106/46 \approx 2.3$ blowup in storage requirement at the server.

Since the key S was chosen from the error distribution and $\|\Xi\|_\infty \leq 1/2$, then the added noise is small and the result is a valid ciphertext. (See more details below.)

Noise Analysis. For the first dimension, we need to use GSW evaluation to compute the encrypted unary vector, where each ciphertext in that vector is a product of $\log N_1 = 8$ ciphertexts. Hence the noise of each these evaluated ciphertexts has roughly the form $\sum_{u=1}^7 E_u \times G_1^{-1}(\text{something})$ with E_u one of the error matrices that were sampled during encryption. Once we multiply by the plaintext matrices for the database to get the compressed encryption as in Eq. (5) and add all the ciphertexts across the N_1 -size dimension, we get a noise term of the form

$$\sum_{v=1}^{N_1} \left(\sum_{u=1}^7 E_u \times G_1^{-1}(\text{something}_u) \right) \times \text{plaintext}_v.$$

(Note that on the right we just multiply by the plaintext matrix whose entries are bounded below 2^{45} , but without any G^{-1} .)¹⁰

The entries of the E_u 's can be chosen from a distribution of variance 8 (which is good enough to avoid the Arora-Ge attacks [5]). The entries of $G^{-1}(\cdot)$ are in the range $[\pm 2]$ (because we have $m_1 = n_1 \log_4(Q)$), so multiplication by $G_1^{-1}(\text{something})$ increases the variance by a factor of less than $2^2 \cdot m_1' \cdot 2^{12} < 2^{21.4}$. Similarly multiplying by a plaintext matrix (of entries in $[\pm 2^{45}]$) increases the variance by a factor of $2^{2 \cdot 45} \cdot n_1 \cdot 2^{12} < 2^{103.6}$. The variance of each noise coordinate is therefore bounded by $2^8 \cdot 7 \cdot 8 \cdot 2^{21.4} \cdot 2^{103.6} < 2^{8+3+3+21.4+103.6} = 2^{139}$. Since each noise coordinate is a weighted sum of the entries of the E_u 's with similar weights, it makes sense to treat it as a normal random variable. A good high probability bound on the size of this error is (say) 16 standard deviations, corresponding to probability $\text{erfc}(16/\sqrt{2}) \approx 2^{-189}$. Namely after folding the first dimension, all the compressed ciphertexts have $\|\text{noise}\|_\infty < 16 \cdot \sqrt{2^{139}} = 2^{73.5}$ with high probability.

As we continue to fold more dimensions, we again multiply the encrypted unary vectors for those dimensions (which are GSW ciphertexts) by the results of the previous dimension (which are compressed ciphertexts) using Eq. (5), this time using G_2 . We note that the GSW ciphertexts in these dimensions are fresh, hence their noise terms are just the matrices E that were chosen during encryption. Thus each of the N_j noise terms in this dimension is of the form $E \times G_2^{-1}(\text{something})$ for one of these E matrices. Moreover, only one of the four terms in each dimension has an encrypted bit $\sigma = 1$ while the other have $\sigma = 0$.

¹⁰ Asymptotically, and disregarding our unconventional way of introducing the plaintexts which optimizes concrete performance, the noise from this step grows linearly with N_1 . If we set $N_1 = O(\log N + \lambda)$ for security parameter λ , the noise from this and the remaining steps will be bounded by $O(\log N + \lambda)$, and so q can be bounded by a constant-degree polynomial of these quantities. Given that the complexity of mod- q multiplication is $\log q \cdot \tilde{O}(\log \log q)$, the asymptotic overhead of our PIR scheme will be $\tilde{O}(\log \log \lambda + \log \log \log N)$.

Hence the term $\sigma \cdot \text{previousNoise}$ appears *only once* in the resulting noise term after folding the j 'th dimension. Therefore folding each small dimension $j \geq 2$ just adds four noise terms of the form $E \times G^{-1}(\text{something})$ to the noise from the previous dimension.

Since G_2 has $m_2 = n_1 \log_{253}(Q)$, then each entry in G_2^{-1} is in the interval $[\pm 2^{52}]$, and multiplying by G_2 increases the variance by a factor of less than $(2^{52})^2 \cdot m_2' \cdot 2^{12} = 3 \cdot 2^{117}$ (recall $m_2' = 6$). With $4(D - 1) = 24$ of these terms, the variance of each coordinate in the added noise term is bounded by $24 \cdot 8 \cdot 3 \cdot 2^{117} = 9 \cdot 2^{123}$. We can therefore use the high-probability bound $16 \cdot \sqrt{9 \cdot 2^{123}} < 2^{67.1}$ on the size of the added noise due to all the small hypercube dimensions.

The analysis so far implies that prior to the modulus switching operation, the noise is bounded in size below $2^{73.5} + 2^{67.1}$. The added noise term due to the rounding error in modulus switching is $S \times \Xi$, and the variance of each noise coordinate in this expression is $8 \cdot n_1' \cdot 2^{12}/2 = 3 \cdot 2^{15}$. Hence we have a high probability bound $16 \cdot \sqrt{3 \cdot 2^{15}} < 2^{12.3}$ on the magnitude of this last noise term. The total noise in the ciphertext returned to the client is therefore bounded by

$$\|\text{noise}\|_\infty < \frac{2^{73.5} + 2^{67.1}}{q'} + 2^{12.3} \approx 2^{13.5} + 2^{7.1} + 2^{12} \approx 2^{14}.$$

Recalling that we use the nearly square gadget matrix H with $p = \sqrt[3]{q} \approx 2^{46/3}$, the noise is indeed bounded below $(p - 1)/2$ as needed, hence the ciphertexts returned to the client will be decrypted correctly with overwhelming probability.

Complexity Analysis. The work of the server while processing the query consists mostly of R_Q multiplications and of FFTs. (The other operations such as additions and applications of $G^{-1}()$ once we did the FFTs take almost no time in comparison.)

With our cyclotomic ring of dimension 2^{12} , each FFT operation is about 10–20 times slower than a ring multiply operation in evaluation representation. But it is easy to see that when N/N_1 times the size L of database entries is large enough, the number of multiplies dwarf the number of FFTs by a lot more than a $20\times$ factor. Indeed, FFTs are only preformed in the initial phase where we process the bits of the index i_i sent by the client (which are independent of L and of N/N_1), and after folding the first dimension (which only applies to $N/N_1 \approx 0.25\%$ of the data). With our settings, the multiplication time should exceed the FFT time once $L \cdot N/N_1$ is more than a few thousands. With $N/N_1 = 4000$ in our example, even holding a single JPEG image in each entry already means that the FFT processing accounts for less than 50% of the overall time. And for movies where $L = 29K$, the FFT time is entirely insignificant.

Let us then evaluate the time spent on multiplications, as a function of the database size. For large $L \cdot N/N_1$, by far the largest number of multiplications is performed when multiplying the GSW ciphertexts by the plaintext matrices encoding the database, while folding the first hypercube dimension. These multiplications have the form $C' := C \times M'H \bmod q'q$ with C' a ciphertext of dimension $n_1 \times n_1$ and $M'H$ a redundant plaintext matrix of dimension $n_1 \times n_2$ (where $n_1 = n_2 = 3$). Using the naïve matrix-multiplication algorithm, we need

$3^3 = 27$ ring multiplications for each of these matrix multiplications, modulo the double-sized modulus $q' \cdot q$. Each ring multiplication (for elements in CRT representation) consists of 2^{12} double-size modular integer multiplication, so each such matrix multiplication takes a total of $2 \cdot 27 \cdot 2^{12} \approx 2^{17.75}$ modular multiplications. For this work, we process a single plaintext matrix, containing about $2^{16.5}$ bytes, so the amortized work is about 2.4 modular multiplication per database byte. (Using Laderman’s method we can multiply 3-by-3 matrices with only 23 multiplications [24], so the amortized work is only 2 modular multiplications per byte.) Taking into account the rest of the work should not change this number in any significant way when L is large, these multiplications likely account for at least 90% of the execution time.

Two (or even three) modular multiplication per byte should be faster than AES encryption of the same data. For example software implementations of AES without any hardware support are estimated at 25 cycles per byte or more [13,32]. Using the fact that we multiply the same GSW matrix by very many plaintext matrices, we may be able to pre-process the modular multiplications, which should make performance competitive even with AES implementations that are built on hardware support in the CPU.

We conclude that for large databases, the approach that we outlined above should be computationally faster than the naïve approach of sending the whole database, even without considering the huge communication savings. We stress that we achieved this speed while still providing great savings on bandwidth, indeed the rate of this solution is 0.44. In other words, compared to the insecure implementation where the client sends the index in the clear, we pay with only $2.25\times$ in bandwidth for obtaining privacy.

Achieving Higher Rate. It is not hard to see that the rate can be made arbitrarily close to one without affecting the asymptotic efficiency. Just before the server returns the answer, it can bootstrap it into another instance of compressible FHE that has rate close to one. This solution is asymptotically cheap, since this bootstrapping is only applied to a single entry. In terms of concrete performance, bootstrapping is very costly so the asymptotic efficiency is only realized for a very large database. Concretely, bootstrapping takes close to 2^{30} cycles per plaintext byte (vs. the procedure above that takes around 2^4 cycles per byte). Hence the asymptotic efficiency is likely to take hold only for databases with at least $N = 2^{30-4} = 64,000,000$ entries.

Acknowledgment. We thank Yuval Ishai for badgering us over the last four years to figure out the achievable rate in LWE-based constructions, until we could bare it no longer and did this work. We also thank Samir Menon and the anonymous reviewers for their useful comments.

References

1. Aguilar-Melchor, C., Barrier, J., Fousse, L., Killijian, M.-O.: XPIR: private information retrieval for everyone. Proc. Priv. Enhancing Technol. **2016**(2), 155–174 (2016)

2. Albrecht, M., et al.: Homomorphic encryption standard, November 2018. <http://homomorphicencryption.org/>. Accessed Feb 2019
3. Angel, S., Chen, H., Laine, K., Setty, S.: PIR with compressed queries and amortized query processing. In: 2018 IEEE Symposium on Security and Privacy (SP), pp. 962–979. IEEE (2018)
4. 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). https://doi.org/10.1007/978-3-642-03356-8_35
5. Arora, S., Ge, R.: New algorithms for learning in presence of errors. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011. LNCS, vol. 6755, pp. 403–415. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22006-7_34
6. Badrinarayanan, S., Garg, S., Ishai, Y., Sahai, A., Wadia, A.: Two-message witness indistinguishability and secure computation in the plain model from new assumptions. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10626, pp. 275–303. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70700-6_10
7. Brakerski, Z., Döttling, N., Garg, S., Malavolta, G.: Leveraging linear decryption: Rate-1 fully-homomorphic encryption and time-lock puzzles. Private communications (2019)
8. Brakerski, Z., Gentry, C., Halevi, S.: Packed ciphertexts in LWE-based homomorphic encryption. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 1–13. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36362-7_1
9. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: Fully homomorphic encryption without bootstrapping. In: Innovations in Theoretical Computer Science (ITCS 2012) (2012). <http://eprint.iacr.org/2011/277>
10. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. *SIAM J. Comput.* **43**(2), 831–871 (2014)
11. Brakerski, Z., Vaikuntanathan, V.: Lattice-based FHE as secure as PKE. In: Naor, M. (ed.) Innovations in Theoretical Computer Science, ITCS 2014, pp. 1–12. ACM (2014)
12. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private information retrieval. In: Proceedings, 36th Annual Symposium on Foundations of Computer Science 1995, pp. 41–50. IEEE (1995)
13. Crypto++ 5.6.0, pentium 4 benchmarks (2009). <https://www.cryptopp.com/benchmarks-p4.html>. Accessed Feb 2019
14. Damgård, I., Jurik, M.: A generalisation, a simplification and some applications of paillier’s probabilistic public-key system. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 119–136. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44586-2_9
15. Döttling, N., Garg, S., Ishai, Y., Malavolta, G., Mour, T., Ostrovsky, R.: Trapdoor hash functions and their applications. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 3–32. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_1
16. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: STOC, pp. 197–206. ACM (2008)
17. 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. LNCS, vol. 8042, pp. 75–92. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_5

18. Green, M., Hohenberger, S., Waters, B.: Outsourcing the decryption of ABE ciphertexts. In: Proceedings 20th USENIX Security Symposium, San Francisco, CA, USA, 8–12 August 2011. USENIX Association (2011)
19. Halevi, S.: Homomorphic encryption. Tutorials on the Foundations of Cryptography. ISC, pp. 219–276. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57048-8_5
20. Hiromasa, R., Abe, M., Okamoto, T.: Packing messages and optimizing bootstrapping in gsw-fhe. IEICE TRANS. Fundam. Electron. Commun. Comput. Sci. **99**(1), 73–82 (2016)
21. Kiayias, A., Leonardos, N., Lipmaa, H., Pavlyk, K., Tang, Q.: Optimal rate private information retrieval from homomorphic encryption. Proc. Priv. Enhancing Technol. **2015**(2), 222–243 (2015)
22. Klinc, D., Hazay, C., Jagmohan, A., Krawczyk, H., Rabin, T.: On compression of data encrypted with block ciphers. IEEE Trans. Inf. Theory **58**(11), 6989–7001 (2012)
23. Kushilevitz, E., Ostrovsky, R.: Replication is not needed: Single database, computationally-private information retrieval. In: Proceedings, 38th Annual Symposium on Foundations of Computer Science 1997, pp. 364–373. IEEE (1997)
24. Laderman, J.D.: A noncommutative algorithm for multiplying 3×3 matrices using 23 multiplications. Bull. Amer. Math. Soc. **82**(1), 126–128 (1976)
25. Lipmaa, H., Pavlyk, K.: A simpler rate-optimal CPIR protocol. In: Kiayias, A. (ed.) FC 2017. LNCS, vol. 10322, pp. 621–638. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70972-7_35
26. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. J. ACM **60**(6), 43 (2013). Early version in EUROCRYPT 2010
27. 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). https://doi.org/10.1007/978-3-642-29011-4_41
28. Olumofin, F., Goldberg, I.: Revisiting the computational practicality of private information retrieval. In: Danezis, G. (ed.) FC 2011. LNCS, vol. 7035, pp. 158–172. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27576-0_13
29. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_16
30. Peikert, C., Vaikuntanathan, V., Waters, B.: A framework for efficient and composable oblivious transfer. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 554–571. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_31
31. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. J. ACM **56**(6), 34:1–34:40 (2009)
32. Schmid, P., Roos, A.: AES-NI performance analyzed; limited to 32nm core i5 CPUs (2010). <https://www.tomshardware.com/reviews/clarkdale-aes-ni-encryption,2538.html>. Accessed Feb 2019
33. Sion, R., Carbunar, B.: On the practicality of private information retrieval. In: Proceedings of the Network and Distributed System Security Symposium, NDSS 2007, San Diego, California, USA, 28 February–2nd March 2007 (2007)
34. Smart, N.P., Vercauteren, F.: Fully homomorphic SIMD operations. Des. Codes Crypt. **71**(1), 57–81 (2014). Early verion at <http://eprint.iacr.org/2011/133>

35. Stern, J.P.: A new and efficient all-or-nothing disclosure of secrets protocol. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 357–371. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-49649-1_28
36. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 24–43. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_2