Compressible FHE with Applications to PIR

Craig Gentry and Shai Halevi

Algorand Foundation*
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 FHE ciphertexts that may have come from a homomorphic evaluation (e.g., of the Gentry-Sahai-Waters (GSW) scheme), 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 faster than whole-database AES encryption – specifically under 1.8 mod-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: Evaluated 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 implementations so far (such as [AMBFK16,ACLS18]) are still quite far from being able to support large databases, mostly because the large expansion factor of contemporary HE schemes. Another application of high-rate HE can be found in the work of Badrinarayanan et al. [BGI+17], 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 [DJ01], 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 [GSW13], 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

* Most of this work was done while the authors were in IBM Research.

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 [Reg09] or its ring variant [LPR13]. (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 under 1.8 single-precision modular multiplications for 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-Carbunar [SC07], 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 [KO97,Ste98] 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 $N = N_1 \times \cdots \times N_d$ down to N/N_1. To maximize

[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 send 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 section 5.

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., [KHJ+12]). In the context of (F)HE, one can view "ciphertext packing" [PVW08,SV14,BGV12,BGH13], 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. [vDGHV10] for integer-based HE, and the work of Hohenberger et al. for attribute-based encryption [GHW11]. 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. [BGI+17].)

The only prior cryptosystem with homomorphic properties that we know of with rate better than 1/2 is due to Damgård and Jurik [DJ01]. They described an extension of the Paillier cryptosystem [Pai99] that allows rate-$(1-o(1))$ encryption with additive homomorphism: In particular, a mod-N^s plaintext can be encrypted inside a 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. [DGI+19] and follow-up work by Brakerski et al. [BDGM19] 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) [CGKS95] 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 [KO97] showed that PIR is possible even with a single server under computational assumptions.

Kiayias et al. [KLL+15] (see also [LP17]) gave the first single-server PIR scheme with rate $(1 - o(1))$, based on Damgård-Jurik [DJ01]. However, Damgård-Jurik is computationally too expensive to be used in practice for large-scale PIR [SC07,OG11], at a minimum, PIR using Damgård-Jurik requires the server to compute a mod-N multiplication per bit of the database, where N has 2048 or more bits. The papers [KLL+15,LP17] expressly call for an underlying encryption scheme to replace Damgård-Jurik to make their rate-optimal PIR schemes computationally less expensive.

[4] In the first step, the server generates N_1 ciphertexts from the clients query, which includes FFTs, but their amortized cost is insignificant when $N_1 \ll N$.

In terms of computation, the state-of-the-art PIR scheme is XPIR by Aguilar-Melchor et al. [AMBFK16], with further optimizations in the SealPIR work of Angel et al. [ACLS18]. 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 [ACLS18, Fig. 9] indicate a rate of roughly 1/1000, compared to our rate of just under 1/2. [5] In terms of server work, SealPIR reports just over twenty cycles per database byte, while the construction here approaches something like 1.8 single-precision modular multiplication per byte (which should be a bit less than 20 cycles). It is noted, however, that our scheme needs entry-size above 100KB to approach top performance, while the performance numbers from [ACLS18] are for 288-byte entries.

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

2 Background on Gadget Matrices, LWE, PVW and GSW

Gadget Matrices. Many lattice cryptosystems (including GSW [GSW13]) use a rectangular *gadget matrix* [MP12], $G \in R_q^{n \times m}$ 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 rationals[6]. 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 $I_{n_1} \otimes \boldsymbol{g}$ where $\boldsymbol{g}$ is the vector $(1, 2, \ldots, 2^{\lfloor \log q \rfloor})$ – that is, $m = n_1 \lceil \log q \rceil$ and G's rows consists of shifts of the vector $\boldsymbol{g}$. In this case, one can efficiently find a suitable $G^{-1}(C)$ that has coefficients in $\{0, 1\}$. More generally with $\boldsymbol{g} = (1, B, \ldots, B^{\lfloor \log_B q \rfloor})$, $G^{-1}(C)$ 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 [Reg09,LPR13]. 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 $\boldsymbol{a}_i \in R_q^k$ together with the values $b_i := \langle \boldsymbol{s}, \boldsymbol{a}_i \rangle + e_i \in R_q$ where $\boldsymbol{s}$ is a fixed secret vector and $e_i \leftarrow \chi$. In a "no" instance, both the $\boldsymbol{a}_i$'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 Figure 2 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 $SP = E \bmod q$. The LWE assumption (matrix version) says that this P is pseudorandom.

[5] The SealPIR rate may be improved significantly for large database entries, but not anywhere near a rate of 1/2.

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

Regev and PVW Encryption. Recall that in the Regev cryptosystem, a bit $\sigma \in \{0,1\}$ is encrypted relative to a secret-key vector $\boldsymbol{s} \in \mathbb{Z}_q^{n+1}$ (with 1 in the last coordinate), as a vector $\boldsymbol{c} \in \mathbb{Z}_q^{n+1}$ such that $\langle \boldsymbol{s}, \boldsymbol{c} \rangle = \lceil q/2 \rfloor \cdot \sigma + e \pmod q$, 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 $\boldsymbol{c} \in R_q^{n+1}$ such that $\langle \boldsymbol{s}, \boldsymbol{c} \rangle = \lceil q/p \rfloor \cdot \sigma + e \pmod q$, 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.)

Peikert et al. described in [PVW08] a batched variant of this cryptosystem (called PVW), where the plaintext is a vector $\boldsymbol{\sigma} \in R_p^k$, the secret key is a matrix $S = (S'|I) \in R_q^{n \times (n+k)}$ and the encryption of $\boldsymbol{\sigma}$ is a vector $\boldsymbol{c} \in R_q^{n+k}$ such that $S \cdot \boldsymbol{c} = \lceil q/p \rfloor \cdot \boldsymbol{\sigma} + \boldsymbol{e}$ with $\|\boldsymbol{e}\|_\infty < q/2p$. 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 $SC = \lceil q/p \rfloor \cdot \Sigma + \boldsymbol{E}$ with $\|\boldsymbol{E}\|_\infty < q/2p$.

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 cryptsystem of Gentry et al. [GSW13], 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 $SP = E \bmod q$ 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

$$S \cdot C = \sigma \cdot S \cdot G + S \cdot P \cdot X = \sigma \cdot S \cdot G + E' \pmod q, \tag{1}$$

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. [HAO16] 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 [Hal17] for detailed treatment of all these variations.)

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

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

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

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

(We allow prefix since the output of $\mathsf{Decrypt}$ *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* $\boldsymbol{b} = (b_1, \ldots, b_t) \in \{0,1\}^t$*, and low rate ciphertexts* $\mathfrak{c}' \leftarrow \mathsf{Evaluate}(\mathsf{pk}, \Pi, \mathsf{Encrypt}(\mathsf{pk}, \boldsymbol{b}))$ *as in eq.* (2) *we have*

$$|\mathsf{Compress}(\mathsf{pk}, \mathfrak{c}')| \cdot \alpha \leq |\Pi(\boldsymbol{b})|.$$

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

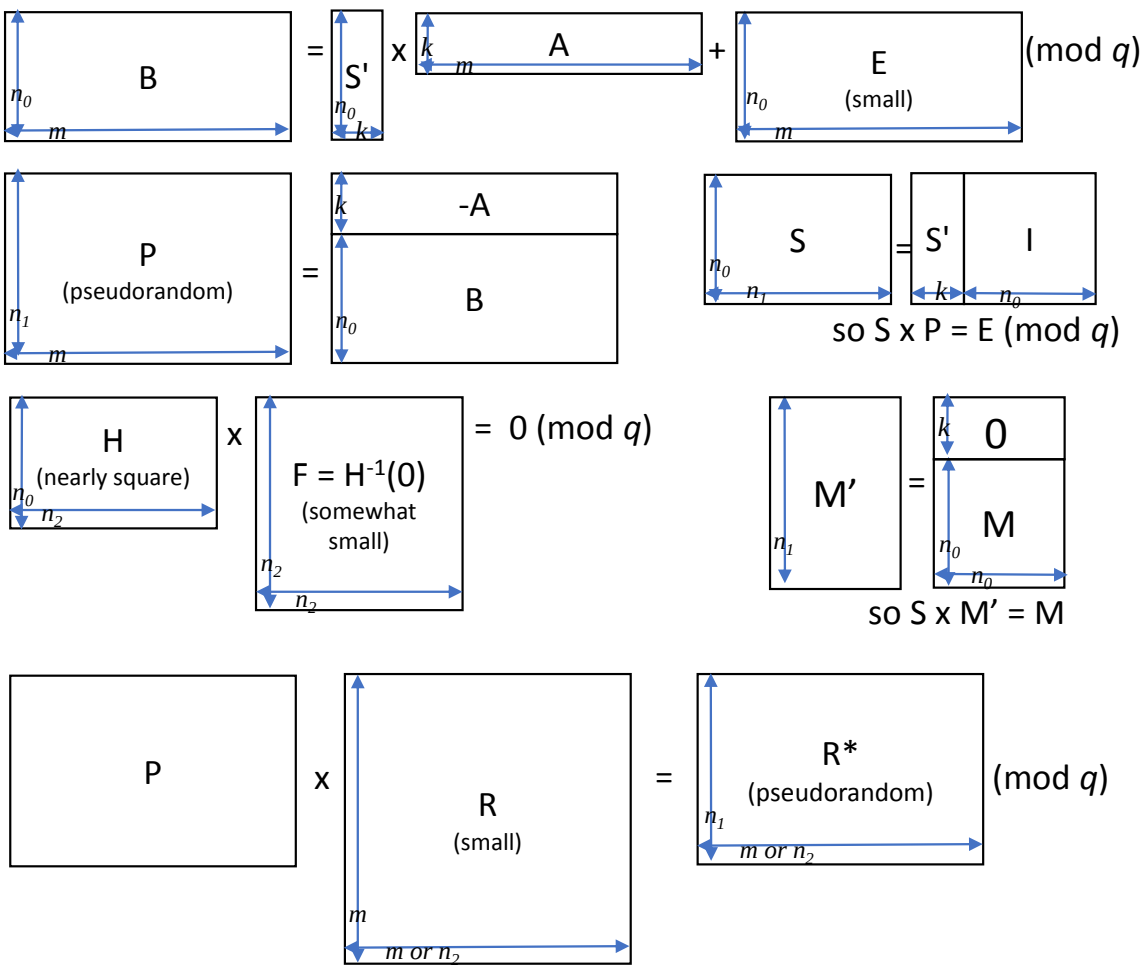

Fig. 1. An illustration of the matrices in our construction. For some small $\epsilon > 0$ we have $n_1 = n_0+k \approx n_2 = n_0(1+\epsilon/2)$ and $m = n_1 \log q$. So, $n_0 \approx 2k/\epsilon$. Also, for correct decryption of ciphertexts with error E using gadget matrix H we require $\|E\|_\infty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

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. [HAO16]. 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 section 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 [PVW08] (in its matrix notations), and the other 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

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 $P = \left[\frac{-A}{B}\right] \in R_q^{n_1 \times m}$, and we have $S \times P = S' \times (-A) + I \times B = E \pmod q$.

Encryption and Evaluation. Encryption of small scalars and homomorphic evaluation are 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 [GSW13].

Evaluation is the same as in GSW, with addition implemented by just adding the ciphertext matrices modulo q and multiplication implemented as $C^\times := C_1 \times G^{-1}(C_2) \bmod q$. 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 $f = \lceil q/p \rfloor$. Also denote $\ell = \lfloor \log p \rfloor$, 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 $\sigma_{i,j,k} \in \{0,1\}$. Namely we have $S \times C_{u,v,w} = \sigma_{u,v,w} SG + E_{u,v,w} \pmod q$ 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 $T_{u,v} = \boldsymbol{e}_u \otimes \boldsymbol{e}_v$ (where $\boldsymbol{e}_u, \boldsymbol{e}_v$ 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, $T'_{u,v} = \left[\frac{0}{\boldsymbol{e}_u \otimes \boldsymbol{e}_v}\right] \in \mathbb{Z}_q^{n_1 \times n_0}$. 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

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

where $Z = [z_{u,v}] \in [p]^{n_0 \times n_0}$. (The equality $(*)$ holds since $S = [S'|I]$ and $T' = \left[\frac{0}{T}\right]$ 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 $Z := \lceil Z/f \rfloor$. 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. Concrete examples (with fully specified constants) are provided in section 5 and appendix A 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 $n_1 = n_0 + k \le n_0(1 + \epsilon/2)$

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 [GSW13,BV14b], 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 equation (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 $\sum_{u,v,w} E_{u,v,w} G^{-1}(\mathsf{something})$, 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 $p^{\epsilon/2} > \mathsf{poly}(k) \log p$. 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 $\mathsf{poly}(k)$ 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 section 5 and appendix A 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 $\mathsf{poly}(\lambda)^{1/\epsilon}$.* □

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 encrypting 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

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

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 $M' = \left[\frac{0}{M}\right] \in \mathbb{Z}_p^{n_1 \times n_0}$, 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 section 4.4), that we call H: Just like the G gadget matrix from [MP12], 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 nearly square 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 $M' = \left[\frac{0}{M}\right]$ 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 section 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 section 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 section 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 [GSW13].

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

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

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

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

where $Z = [z_{u,v}] \in [q]^{n_0 \times n_0}$. (The equality $(*)$ holds since $S = [S'|I]$ and $T' = \left[\frac{0}{T}\right]$ 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 $X := S\ C = M\ H + E \pmod q$ 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 section 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 [GSW13,BV14b], if we use fresh-ciphertext noise of size $\mathsf{poly}(k)$ then also after bootstrapping we still have the noise magnitude bounded below $\mathsf{poly}(k)$. After compression as per Eqn. (6), the noise term is a sum of $n_0^2 \log q$ matrices $E_{u,v,w}$, all of magnitude bounded by $\mathsf{poly}(k)$, hence it has magnitude below $\mathsf{poly}(k) \cdot \log q$. 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 section 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 $\frac{q^{\epsilon/2}}{2} > \mathsf{poly}(k) \log q$ (essentially the same as for the variant from section 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 [MP12] $G \in \mathbb{Z}_q^{n_1 \times m}$ 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 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 $G = I \otimes \boldsymbol{g}$ where $\boldsymbol{g} = (1, B, \ldots, B^{\lfloor \log_B q \rfloor})$ 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 $\boldsymbol{g}$ must have dimension at least 2. Achieving 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. $G \cdot F = 0 \bmod q$
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. [GPV08].

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:

$$F' := \begin{bmatrix} p^{t-1} & p^{t-2} & \cdots & p & 1 \\ 1 & p^{t-1} & \cdots & p^2 & p \\ p & 1 & \cdots & p^3 & p^2 \\ & \vdots & & \ddots & \vdots \\ p^{t-2} & p^{t-3} & \cdots & 1 & p^{t-1} \end{bmatrix} \tag{7}$$

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

$$\|\boldsymbol{v}F'\|_\infty \leq \|\boldsymbol{v}\|_\infty \cdot (1+p+\ldots+p^{t-1}) = \|\boldsymbol{v}\|_\infty \cdot (p^t-1)/(p-1) = \|\boldsymbol{v}\|_\infty \cdot \tfrac{q}{p-1}. \tag{8}$$

Moreover this bound is rather tight, in particular $\|\boldsymbol{v}F'\|_\infty > \|\boldsymbol{v}\|_\infty \cdot \frac{q}{p-1} \cdot (1-\frac{2}{p})$.

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 $n_2 = rt = n_0(1+\frac{1}{t-1})$.

When decrypting compressed ciphertexts, 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 $(ZF \bmod q) = EF$ over the integers. Now we use the fact that F has full rank over the reals, and recover $E := (ZF \bmod q) \times F^{-1}$. 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.

Computing $\boldsymbol{H^{-1}(x)}$ for arbitrary vectors $\boldsymbol{x}$. While we do not use it in this work, it may still be useful in other applications to find $H^{-1}(\boldsymbol{x})$ for things other then the zero vector. Namely compute $\boldsymbol{y} = H^{-1}(\boldsymbol{x})$ of norm $\ll q$ such that $H\boldsymbol{y} = \boldsymbol{x} \pmod q$. Such a procedure can be used for computing $Y = H^{-1}(X)$ for a matrix $X \in \mathbb{Z}_q^{r \times n_0}$, just by applying it to each column of X. Consider for simplicity the basic matrix F' from eq. (7) and the matrix $H \in \mathbb{Z}_q^{(t-1)\times t}$ whose rows span the orthogonal space to the columns of F' modulo q. (The extension to $F' \otimes I$ and $H \otimes I$ is obvious.)

Computing $\boldsymbol{y} = H^{-1}(\boldsymbol{x})$, is done by first finding an arbitrary vector $\boldsymbol{y'} \in \mathbb{Z}_q^t$ satisfying $H\boldsymbol{y'} = \boldsymbol{x} \pmod q$, and then reducing $\boldsymbol{y'}$ modulo the lattice spanned by the columns of F'. There always exists a vector $\boldsymbol{y'}$ as needed (since H has rank $t-1$), so it is left to show how to reduce it modulo $L(F')$ and bound the size of the result. As the diagonal of F' (which is p^{t-1}) is much larger than all other entries, the following lemma essentially says that all we need to do is reduce each entry y'_i modulo the corresponding diagonal entry of F'.

Lemma 3. *Consider an arbitrary vector $\boldsymbol{z} \in \mathbb{Z}^t$, and let $\boldsymbol{z'} = \boldsymbol{z} - F' \cdot \boldsymbol{w}$ where F' is the matrix from eq. (7) and $\boldsymbol{w} = \lceil \boldsymbol{z}/p^{t-1} \rfloor$. Then $\|\boldsymbol{z'}\|_\infty \leq \frac{p^{t-1}}{2}\big(1+\frac{1}{p-1}\big) + \frac{\|\boldsymbol{z}\|_\infty}{p-1}$.*

Proof. Denote the i'th row in F' by $\boldsymbol{f}^i = (p^{i-1}, p^{i-2}, \ldots, p^i)$, (indexing starts from zero and the exponent computed modulo t). Note in particular that $\boldsymbol{f}^i_i = p^{t-1}$. The i'th entry in the reduced vector is

$$\boldsymbol{z}'_i \;=\; \boldsymbol{z}_i - \sum_{j=0}^{t-1} \boldsymbol{f}^i_j \boldsymbol{w}_j \;=\; \boldsymbol{z}_i - p^{t-1}\boldsymbol{w}_i - \sum_{j \neq i} \boldsymbol{f}^i_j \boldsymbol{w}_j \;=\; (\boldsymbol{z}_i \bmod p^{t-1}) - \sum_{j \neq i} \boldsymbol{f}^i_j \boldsymbol{w}_j.$$

The last equality above follows because $\boldsymbol{w}_i = \lceil \boldsymbol{z}_i/p^{t-1} \rfloor$, and clearly $|\boldsymbol{z}_i \bmod p^{t-1}| \leq p^{t-1}/2$.

As $\boldsymbol{w} = \lceil \boldsymbol{z}_i/p^{t-1} \rfloor$, then all the $\boldsymbol{w}_j$'s are bounded in magnitude by $\beta = \|\boldsymbol{z}\|_\infty/p^{t-1} + \frac{1}{2}$. The term $\sum_{j\neq i} \boldsymbol{f}_j^i \boldsymbol{w}_j$ is therefore an inner product between the geometric series $(1, p, \ldots, p^{t-2})$ and a β-bounded vector, so its magnitude is bounded by

$$|\sum_{j\neq i} \boldsymbol{f}_j^i \boldsymbol{w}_j| \;\leq\; \beta(1+p+\ldots+p^{t-2}) \;<\; \left(\frac{\|\boldsymbol{z}\|_\infty}{p^{t-1}} + \frac{1}{2}\right)\cdot \frac{p^{t-1}-1}{p-1} \;<\; \frac{\|\boldsymbol{z}\|_\infty}{p-1} + \frac{p^{t-1}}{2(p-1)}.$$

The overall magnitude of $\boldsymbol{z}'_i$ is therefore bounded by

$$|\boldsymbol{z}'_i| \;<\; \frac{p^{t-1}}{2} + \frac{\|\boldsymbol{z}\|_\infty}{p-1} + \frac{p^{t-1}}{2(p-1)} \;\leq\; \frac{p^{t-1}}{2}\big(1+\frac{1}{p-1}\big) + \frac{\|\boldsymbol{z}\|_\infty}{p-1}$$

□

Applying lemma 3 to $\boldsymbol{y'} \in [\pm\frac{q}{2}]$ yields a reduced vector $\boldsymbol{y}$ with entries bounded by just over p^{t-1}. We can reduce it again (using the same lemma) and get an even smaller vector with entries just slightly over $p^{t-1}/2$. Moreover since $\boldsymbol{y} = \boldsymbol{y'} - F' \cdot \boldsymbol{w}$ (for some $\boldsymbol{w}$) then

$$H\boldsymbol{y} = H(\boldsymbol{y'} - F'\boldsymbol{w}) = H\boldsymbol{y'} - HF'\boldsymbol{w} = \boldsymbol{x} \pmod q.$$

More general moduli q. 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 $\boldsymbol{u} = (1, a, \cdots, a^{t-1})$ modulo q, where $a = \lceil q^{1/t} \rceil$. Let the rows of F' be the L-vectors $c_i \cdot \boldsymbol{u} \bmod q$ 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 $c_i \cdot \boldsymbol{u} \bmod q$, which is $\lceil q/a^i \rceil \cdot a^j \bmod q$ for $i \in [t]$, $j \in \{0, \ldots, t-1\}$. 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 $\lceil q/a^i \rceil \cdot a^j \bmod q$ 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.

The procedure from above for computing $H^{-1}(\boldsymbol{x})$ will not quite work out the box anymore, but generic techniques for using $H^{-1}(0)$ to compute $H^{-1}(\boldsymbol{x}$ yield similar results (with slightly worse bounds).

5 Application to Fast Private Information Retrieval

Private information retrieval (PIR) lets clients retrieve entries from a database held by a server without the server learning what entries were retrieved. A naïve solution would have the server send its entire database to the client. PIR protocols [CGKS95] offer ways to reduce the bandwidth overhead as compared to this naïve solution, making it sublinear in the number N of entries in the database. It is well known that homomorphic encryption can be used for PIR: The client can encrypt the index of the entry that it wants, and the server can perform homomorphic evaluation of a table lookup function, returning the encrypted entry. This solution has the server returning a

single encrypted entry, so the bandwidth overhead is mostly just the plaintext-to-ciphertext ratio of the homomorphic scheme, which is independent of the number of entries. But if the entries themselves are large (as in a movie database where each movie is several gigabytes long), then even this N-independent overhead could be very costly. See [KLL+15,AMBFK16,LP17] for more motivating discussions. Clearly, if we have a compressible HE scheme we could use it to make the bandwidth overhead arbitrarily close to one.

The first single-server rate-1 PIR scheme for large entries was described by Kiayias et al. [KLL+15], using the Damgård-Jurik encryption scheme [DJ01] that supports plaintext-to-ciphertext expansion ratio arbitrary close to one. But Damgård-Jurik is only additively homomorphic, so their solution becomes much more complicated, relying on the Ishai-Paskin homomorphic construction for branching-programs [IP07] (which internally uses Damgård-Jurik). Our compressible HE schemes offers a new avenue for getting a rate-1 PIR, relying on the (quantum-safe) LWE hardness rather than the N-th residuosity assumption of Damgård-Jurik.

Computational efficiency is also a major issue with PIR, any PIR scheme must touch all the bits in the database for every query and most single-server PIR schemes apply a rather expensive processing for each bit. In particular the rate-optimal scheme from [KLL+15] must apply at a minimum one multiplication (modulo an RSA modulus of at least 2048 bits) for every bit in the database. This was highlighted by a study of Sion and Carbunar [SC07], who concluded (in 2007) that "*deployment of non-trivial single server PIR protocols on real hardware of the recent past would have been orders of magnitude less time-efficient than trivially transferring the entire database.*" We note that sending the entire database is not computationally free: it involved at least encrypting the whole database for purposes of communication security. Still, over a decade after the Sion-Carbunar study, prior to our work we still did not have any single-server PIR scheme that can compete with the trivial scheme in terms of computation. Below we describe a series of optimizations for our scheme, yielding a construction which is not only bandwidth efficient but should also outperform whole-database AES encryption.[7] In appendix A we describe yet another variant with much smaller query size and even somewhat faster server processing. (Specifically the variant here uses about 2.3 multiplies per database byte while the one in appendix A uses about 1.8 multiplies per byte.) (Specifically the variant here uses about 2.3 multiplies per database byte while the one in appendix A uses about 1.8 multiplies per byte.)

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 $N = N_1 \times \cdots \times N_D$ and the scheme uses degree-D homomorphism:

- The client's index $i \in [N]$ is represented in mixed radix of basis $N_1, \ldots, N_D$, namely as $(i_1, \ldots, i_D)$ 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 $N/N_1 = N_2 \times \ldots N_D$, consisting only the i_1'st hyper-row of the database.

[7] The "should" is since we did not implement this construction. Implementing it and measuring its performance may be an interesting topic for future work.

– 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 from section 4.2 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 (and $n_2, n_1 > n_0$), 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 $C \times M$ 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.[8]

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.)

[8] 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.

As in the XPIR work [AMBFK16], 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.

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 $N_2 = N_3 = \cdots = N_D = 4$. 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 (easily $N = 2^{100}$).

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 $N_2 = N_3 = \cdots = 4$, we have the client directly sending the encrypted unary vectors for these dimensions. Namely for each $j = 2, 3, \ldots$ the client sends four ciphertexts $C_{j,0}, \ldots, C_{j,3}$ 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 $\log N_1 + (N_2 + N_3 + \cdots N_D)$ 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 $E \times G^{-1}(\mathsf{something})$, 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 $E \times G^{-1}(\mathsf{something}) \times G^{-1}(\mathsf{something}')$, 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 $G^{-1}(\mathsf{something})$ 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 [BV14a,BGV12]. 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.

Our initial noise analysis indicated that setting the GSW G matrix to be very wide and short would have let us keep the noise very small throughout the computation, hence allowing us to use a modulus q of roughly 64 bits. But this would have entailed a large increase in bandwidth, computation and storage at the server. For example the server needs to store in CRT basis all the plaintext matrices $G^{-1}(M')$ which is more than 32 times larger than the plaintext matrix M itself. (Also we could not find parameters that would allow us to achieve rate better than 1/3 for that case.)

A better option is to use a significantly larger modulus, at least initially, and then use modulus switching to scale it down to 64 bits (or perhaps even smaller). 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 [ACPS09]. (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. (Consulting Table 1 from [ACC+18], 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 section 4.4 with a 2-by-3 matrix H.

We split a size-N database into a hypercube of dimensions $N = 256 \times 4 \times 4 \times \ldots \times 4$. A client wishing to retrieve an entry $i \in [N]$ first represents i as $(i_1, i_2, \ldots, i_D)$, with $i_i \in [256]$ and $i_j \in [4]$ for all $j > 1$. Let $\sigma_{1,0}, \ldots \sigma_{1,7}$ be the bits of i_1, the client then encrypts the scalars $q' \cdot \sigma_{1,0}$ and $\sigma_{1,1}, \ldots, \sigma_{1,7}$ in GSW ciphertexts (modulo Q). For $j = 2, \ldots, D$ 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 $C_{1,0} \in R_Q^{n'_1 \times n'_1}$ 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 $C_{1,t} \in R^{n'_1 \times m'_1}$ 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 198MB).

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. Moreover, in the appendix we describe a variation of the scheme from here that uses the query-expansion technique in the SealPIR work [ACLS18]. That scheme is even slightly more computationally efficient than the one we describe here, and while its total query-size is even more than the one here, most of that query consists of key material, independent of the client query. The "online" portion of the query is under 30MB, a saving of about 85%.

The server pre-processes its database by breaking each entry into 2-by-2 plaintext matrices $M \in R_q$ (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 compute $M'' = M'H \pmod q$ and encodes each entry in the M'' 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 4GB movie will be encoded in about 44K 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 $r = 1, 2, \ldots, 256$ the server multiplies the r'th ciphertext in this vector by all the plaintext matrices M'' 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 $N_2 \times \cdots \times N_D$), 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 $C' = \lceil C/q' \rfloor \in R_q$, 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

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

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 $\sum_{u=1}^{7} E_u \times G_1^{-1}(\mathsf{something})$ with E_u one of the error matrices that were sampled during encryption. Once we multiply by the plaintext

[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 $106/46 \cdot 3/2 \approx 3.5$ blowup in storage requirement at the server.

matrices for the database to get the compressed encryption as in Equation (5) and add all the ciphertexts across the N_1-size dimension, we get a noise term of the form

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

(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 [AG11]). 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 $G_1^{-1}(\mathsf{something})$ increases the variance by a factor of less than $2^2 \cdot m'_1 \cdot 2^{12} = 159 \cdot 2^{14}$. Similarly multiplying by a plaintext matrix M'' over R_q (of entries in $[\pm 2^{45}]$) increases the variance by a factor of $2^{2 \cdot 45} \cdot n'_1 \cdot 2^{12} = 3 \cdot 2^{102}$. The variance of each noise coordinate is therefore bounded by $2^8 \cdot 7 \cdot 8 \cdot 159 \cdot 2^{14} \cdot 3 \cdot 2^{102} = 3339 \cdot 2^{127}$.

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 Equation (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 $E \times G_2^{-1}(\mathsf{something})$ 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 $\sigma = 0$. Hence the term $\sigma \cdot \mathsf{previousNoise}$ 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 $E \times G^{-1}(\mathsf{something})$ to the noise from the previous dimension.

Since G_2 has $m_2 = n_1 \log_{2^{53}}(Q)$, 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 $24 \cdot 8 \cdot 3 \cdot 2^{117} = 9 \cdot 2^{123}$. Hence the noise variance after folding all the dimensions is bounded below $3339 \cdot 2^{127} + 9 \cdot 2^{123} < 3340 \cdot 2^{127}$.

The server then applies modulus switching, scaling down the noise by a factor of $q' \approx 2^{60}$ (and hence the variance is scaled sown by a factor of 2^{120}). On the other hand, modulus switching also adds a noise term due to rounding of the form $S \times \Xi$ (with S the secret key and Ξ the rounding error). The variance of each noise coordinate in this last expression is $8 \cdot n'_1 \cdot 2^{12}/2 = 3 \cdot 2^{15}$. Therefore, the total noise variance after modulus switching is bounded below $3340 \cdot 2^7 + 3 \cdot 2^{15} = 525824 \approx 2^{19}$.

Since each noise coordinate is a weighted sum of may noise terms which are zero-mean with similar variance, it makes sense to treat it as a zero-mean normal random variable with parameter $\sigma \approx 2^{9.5}$. Recalling that we use the nearly square gadget matrix H with $p = \sqrt[3]{q} \approx 2^{46/3}$, the noise magnitude must be bounded below $\beta = (p-1)/2 \approx 2^{14.3}$ to be able to decrypt. This correspond to about $2^{14.3}/2^{9.5} \approx 28$ standard deviations, so the probability of any one noise coeeficient begin larger that this bound β is less than $\mathsf{erfc}(28/\sqrt{2}) < 2^{-570}$. Even using the union bound and multiplying by the number of variables, the error probability remains completeley negligible.

[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 $\tilde{O}(\log \log \lambda + \log \log \log N)$.

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 [Lad76], 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 [SR10,Cry09]. 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 TCC reviewers for their useful comments.

References

[ACC+18] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic encryption standard. Availabale at `http://homomorphicencryption.org/`, accessed February 2019, November 2018.

[ACLS18] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. PIR with compressed queries and amortized query processing. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 962–979. IEEE, 2018.

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, pages 595–618, 2009.

[AG11] Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In *ICALP (1)*, volume 6755 of *Lecture Notes in Computer Science*, pages 403–415. Springer, 2011.

[AMBFK16] Carlos Aguilar-Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. XPIR: Private information retrieval for everyone. *Proceedings on Privacy Enhancing Technologies*, 2016(2):155–174, 2016.

[BDGM19] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Leveraging linear decryption: Rate-1 fully-homomorphic encryption and time-lock puzzles. Private communications, 2019.

[BGH13] Zvika Brakerski, Craig Gentry, and Shai Halevi. Packed ciphertexts in LWE-based homomorphic encryption. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *Public Key Cryptography*, volume 7778 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2013.

[BGI+17] Saikrishna Badrinarayanan, Sanjam Garg, Yuval Ishai, Amit Sahai, and Akshay Wadia. Two-message witness indistinguishability and secure computation in the plain model from new assumptions. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 275–303. Springer, 2017.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science (ITCS'12)*, 2012. Available at `http://eprint.iacr.org/2011/277`.

[BV14a] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. *SIAM Journal on Computing*, 43(2):831–871, 2014.

[BV14b] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based FHE as secure as PKE. In Moni Naor, editor, *Innovations in Theoretical Computer Science, ITCS'14*, pages 1–12. ACM, 2014.

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 41–50. IEEE, 1995.

[Cry09] Crypto++ 5.6.0, pentium 4 benchmarks. `https://www.cryptopp.com/benchmarks-p4.html`, accessed February 2019, 2009.

[DGI+19] Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. Trapdoor hash functions and their applications. In *"Advances in Cryptology - CRYPTO 2019"*, Lecture Notes in Computer Science. Springer, 2019.

[DJ01] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *International Workshop on Public Key Cryptography*, pages 119–136. Springer, 2001.

[GHS12] Craig Gentry, Shai Halevi, and Nigel Smart. Homomorphic evaluation of the AES circuit. In *"Advances in Cryptology - CRYPTO 2012"*, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867. Springer, 2012. Full version at `http://eprint.iacr.org/2012/099`.

[GHW11] Matthew Green, Susan Hohenberger, and Brent Waters. Outsourcing the decryption of ABE ciphertexts. In *20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings*. USENIX Association, 2011.

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

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013, Part I*, pages 75–92. Springer, 2013.

[Hal17] Shai Halevi. Homomorphic encryption. In *Tutorials on the Foundations of Cryptography*, pages 219–276. Springer International Publishing, 2017.

[HAO16] Ryo Hiromasa, Masayuki Abe, and Tatsuaki Okamoto. Packing messages and optimizing bootstrapping in gsw-fhe. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, 99(1):73–82, 2016.

[HS14] Shai Halevi and Victor Shoup. Algorithms in HElib. In *CRYPTO (1)*, volume 8616 of *Lecture Notes in Computer Science*, pages 554–571. Springer, 2014. Long version in `https://eprint.iacr.org/2014/106`.

[IP07] Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In *Theory of Cryptography Conference*, pages 575–594. Springer, 2007.

[KHJ+12] Demijan Klinc, Carmit Hazay, Ashish Jagmohan, Hugo Krawczyk, and Tal Rabin. On compression of data encrypted with block ciphers. *IEEE transactions on information theory*, 58(11):6989–7001, 2012.

[KLL+15] Aggelos Kiayias, Nikos Leonardos, Helger Lipmaa, Kateryna Pavlyk, and Qiang Tang. Optimal rate private information retrieval from homomorphic encryption. *Proceedings on Privacy Enhancing Technologies*, 2015(2):222–243, 2015.

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 364–373. IEEE, 1997.

[Lad76] Julian D. Laderman. A noncommutative algorithm for multiplying 3×3 matrices using 23 multiplications. *Bull. Amer. Math. Soc.*, 82(1):126–128, 01 1976.

[LP17] Helger Lipmaa and Kateryna Pavlyk. A simpler rate-optimal CPIR protocol. In *International Conference on Financial Cryptography and Data Security*, pages 621–638. Springer, 2017.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6):43, 2013. Early version in EUROCRYPT 2010.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 700–718. Springer, 2012.

[OG11] Femi Olumofin and Ian Goldberg. Revisiting the computational practicality of private information retrieval. In *International Conference on Financial Cryptography and Data Security*, pages 158–172. Springer, 2011.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238. Springer, 1999.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *Advances in Cryptology - CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 554–571. Springer, 2008.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):34:1–34:40, 2009.

[SC07] Radu Sion and Bogdan Carbunar. On the practicality of private information retrieval. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2007, San Diego, California, USA, 28th February - 2nd March 2007*, 2007.

[SR10] Patrick Schmid and Achim Roos. AES-NI performance analyzed; limited to 32nm core i5 CPUs. `https://www.tomshardware.com/reviews/clarkdale-aes-ni-encryption,2538.html`, accessed February 2019, 2010.

[Ste98] Julien P Stern. A new and efficient all-or-nothing disclosure of secrets protocol. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 357–371. Springer, 1998.

[SV14] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *Des. Codes Cryptography*, 71(1):57–81, 2014. Early verion at `http://eprint.iacr.org/2011/133`.

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, pages 24–43, 2010.

A Efficient PIR with Shorter Queries

A serious drawback of the PIR scheme from section 5 is the large query size, nearly 200M for every query. To do better, we adapt the query-expansion technique from SealPIR [ACLS18] (which can be thought of as a specialized variant of the Halevi-Shoup replication algorithm from [HS14, Sec. 4.2]). That technique lets the client send just a single ciphertext encrypting the integer $i_1 \in [N_1]$ (in some particular representation), and the server can efficiently turn this into N_1 ciphertexts encrypting the bits of the i_1'st unit vector in dimension N_1, as needed for the PIR scheme. While we cannot directly apply this technique to our scheme from section 5, we show below how to modify that scheme to make the expansion trick applicable.

Query expansion can be thought of as using packed ciphertexts, encrypting elements of a large algebraic ring that encode vectors of plaintext values (rather than a single value per ciphertext). The client encodes the unit vector in a single ring element and encrypts that element, and the server performs homomorphic unpacking, resulting in N_1 ciphertexts each encrypting a single entry of the vector. Importantly for our case, unpacking *does not involve homomorphic multiplications*, rather it only uses additions, multiplication by plaintext scalars, and another operation called *automorphism*, corresponding roughly to a cyclic shift of the encrypted vector.

While Regev-type cryptosystems support efficient homomorphic automorphism, this is not the case for GSW. In fact the only way that we know of implementing automorphisms in GSW uses bootstrapping. Below we will therefore ditch GSW encryption for the first index $i_1 \in [N_1]$ in our PIR scheme, and replace it with Regev encryption. We note that while we can use Regev-type cryptosystem for encrypting the first index i_1, we still need to encrypt the other indexes $i_2, i_3, \ldots$ using GSW. The reason is that the first encrypted index is multiplied by the plaintext database, while the later indexes are multiplied by ciphertexts.

The efficient automorphism procedures for Regev-type cryptosystems rely on having *key-switching gadgets* in the public key, where every "rotation amount" needs its own key-switching gadget, of size at least 2-3 ciphertexts. These gadgets must be sent to the server, but they do not depend on the client query and can be installed at the server during some setup phase. Once installed, they enable performing many PIR instances with shorter queries.

The rest of this section is organized as follows: we first describe the automorphism operation and its homomorphic implementation for the Regev-type cryptosystem that we use, then describe our variant of the query-expansion technique, and finally the PIR scheme that uses it.

A.1 Automorphisms

In the cyclotomic rings that we use in our homomorphic cryptosystem, the ring elements are polynomials modulo some cyclotomic $\Phi_m(X)$, and the ring automorphisms are mappings of the form $\tau_t : a(X) \mapsto a(X^t) \bmod \Phi_m(X)$ for some integer t (which we call the *automorphism amount*). These automorphisms are mappings from the ring to itself, they commute with ring addition and multiplication, and they all fix the base ring $\mathbb{Z}$. (That is, we have $\tau_t(n) = n$ for every $n, t \in \mathbb{Z}$.) These automorphisms on R imply the same operations on R_q (for any q), below we somewhat abuse notations and denote both types of automorphisms by the same τ_t.

Regev Cryptosystem and its Homomorphic Automorphisms. To get efficient homomorphic implementation of automorphisms, we use a redundant variant of RLWE-based Regev encryption. Specifically, we replace the GSW decryption invariant $S \times C = \sigma S \times G + E \pmod{Q}$ by the decryption invariant[11]

$$S \times C = \sigma G + E \pmod{Q}.$$

In our setting the encryption is performed by the client that holds the secret key $S = (S'|I) \in R_Q^{n_0' \times n_1'}$. To encrypt a scalar σ the client choose a pseudorandom matrix $P \in R_Q^{n_1' \times m'}$ such that $SP = E \pmod{Q}$ for a low-norm E, then outputs the ciphertext $C = P + \sigma \cdot (\frac{0}{G}) \bmod Q$.

Like every Regev cryptosystem, we get additive homomorphism for free as long as the noise is small enough. Implementing automorphisms for this cryptosystem is done in three steps:

- For each automorphism amount t that we want to support, we add a key-switching gadget to the public key. Let $S_t = \tau_t(S)$ be the secret-key matrix obtained by applying the automorphism τ_t to each element of S. The key-switching gadget for this t is a matrix W_t satisfying

  $$S \times W_t = S_t \times \hat{G} + E_t \pmod{q}$$

  for a low norm E_t. (The gadget $\hat{G}$ is different from G above, in that it has n_1' rows rather than n_0. Also note that including this W_t in the public key relies on some circular security assumption.)
- Given a ciphertext matrix C over R_Q, we apply the ring automorphism τ_t to each ring element in C, obtaining a new ciphertext $C_t = \tau_t(C)$. Then we have

  $$S_t \times C_t = \tau_t(S) \times \tau_t(C) = \tau_t(\sigma) \cdot \tau_t(G) + \tau_t(E) = \tau_t(\sigma) \cdot G + \tau_t(E) \pmod{Q}$$

  where the last equality holds since τ_t fixes the elements of the base ring $\mathbb{Z}_q$ and G only consists of integers. Moreover, in the cyclotomic rings that we use, the automorphisms preserve the low norm of the noise, in particular the canonical-embedding norm of $\tau_t(E)$ is the same as that of E itself. It follows that $C_t = \tau_t(C)$ is a valid encryption of the ring element $\tau_t(\sigma)$ relative to the secret key $\tau_t(S)$.
- Finally, we perform key-switching by setting $C' = W \times \hat{G}^{-1}(C_t) \pmod{Q}$. We claim that C' is a valid encryption of $\tau_t(\sigma)$ relative to the original key S since

  $$\begin{aligned} SC' = SW\hat{G}^{-1}(C_t) \;&=\; (S_t\hat{G} + E_t)\hat{G}^{-1}(C_t) \;=\; S_tC_t + E_t\hat{G}^{-1}(C_t) \\ &= \tau(\sigma) \cdot G + \underbrace{\tau_t(E) + E_t\hat{G}^{-1}(C_t)}_{E'} \pmod{Q}. \end{aligned}$$

  The noise term E' of C' consists of $\tau_t(E)$ (that has the same norm as E) and the added term $E_t \times \hat{G}^{-1}(C_t)$. [12]

A.2 Query Expansion

A "generic" unpacking operation using additions, multiplication-by-plaintext-scalars, and rotations, was described by Halevi and Shoup in [HS14, Sec. 4.2]. Angel et al. later described in [ACLS18] a

[11] The G matrix in the new invariant has only n_0' rows, vs. the n_1' rows of the G matrix in GSW.

[12] Other variants are also possible, [GHS12] describes a key-switching procedure where the added noise term has the form $E \times C/Q' + S \times E'$ where Q' is a parameter, S is the secret key, and E' is a rounding-error term.

more algebraic variant, specialized to the case of unit vectors over power-of-two cyclotomic rings. These procedures are very similar, both employing a shift-and-add approach with multiplication by some plaintext scalars. The main difference is that the procedure from [ACLS18] only multiplies by special plaintext scalars that do not increase the noise.

The procedure of Angel et al. begins with (an encryption of) a single ring element σ, using $\sigma = X^{i_1} \in R$ to encode an index $i_1 \in [N_1]$. It consists of $\ell = \log N_1$ iterations, each doubling the number of ring elements while maintaining the invariant that exactly one of them is nonzero. Specifically, after iteration j we have a list of 2^{j+1} elements, where $\mathsf{list}[k] = X^{i_1-k}$ if $k = i_1 \pmod{2^j}$ and $\mathsf{list}[k] = 0$ otherwise. The pseudo-code in fig. 2, adapted from [ACLS18, Fig. 3], describes our variant of this procedure. (The only difference from [ACLS18] is the division-by-2 in Lines 7,8.)

```
Expand(σ = X^{i_1} ∈ R) :  // i_1 ∈ {0,...,2^ℓ − 1}
 1  list = [σ]  // a size-1 list
 2  for j = 0 to ℓ − 1 do  // each iteration doubles the list size
 3      t = M/2^j + 1        // the automorphism amount, M is the ring dimension
 4      for k = 0 to 2^j − 1 do  // double each current element
 5          σ_0 ← list[k]
 6          σ_1 ← σ_0 · X^{−2^j} ∈ R
 7          list[k]   ←     (σ_0 + τ_t(σ_0))/2
 8          list[k + 2^j] ← (σ_1 + τ_t(σ_1))/2
 9  return list
```

Fig. 2. A variant of the query-expansion procedure from [ACLS18]

It is easy to verify that this procedure satisfies the invariant from above, see [ACLS18, Thm. 1]. This mean in particular that at the end we have a list of 2^ℓ elements, the i_1'st of which is one and all the others are zero.

In terms of homomorphic evaluation of this procedure, we still need to explain how to implement the division-by-2 operation in Lines 7,8 and to analyze the noise. We will do both later in this section, but first let us mention a few tradeoffs that we leave out of the current exposition, but that could perhaps be useful for actual implementations.

Some tradeoffs. The procedure above uses ℓ different automorphism amounts, namely $t = 1, 2, \ldots, 2^{\ell-1}$. For each of these amounts we will need a key-switching gadget and these gadgets have significant size. It is possible to use less automorphism amounts, paying with increased time and depth. This is done by noting at $\tau_{2t}(a)$ can always be implemented as $\tau_t(\tau_t(a))$. So if we have a key-switching gadget for t, we do not strictly need also a gadget for $2t$. In the extreme, we can use only a single gadget for $t = 1$, at the price of having to performs $O(N_1 \log N_1)$ operations with computation depth of $O(N_1)$. More generally, we can use $\log N_1/x$ different amounts and pay a factor of x in the number of operations that we need to perform and a factor of $O(2^x)$ in the depth.

In the other direction, Halevi and Shoup described in [HS14] a variant that still uses $O(N_1)$ operations but only has depth $O(\log \log N_1)$, basically "flattening" the top levels of the tree and leaving only the bottom $\log \log N_1$ levels. This modification can be applies also to the procedure from [ACLS18]. This procedure, however, uses $O(\frac{N_1}{\log N_1})$ different automorphism amounts.

A.3 PIR Using Query Expansion

We now describe our efficient PIR construction that uses all the components from above. As we already mentioned, we will use the redundant Regev cryptosystem to encrypt the first index i_1 (with the server running a homomorphic query expansion), but GSW to encrypt all the other indexes i_j.

Parameters. Let $q \approx 2^{42}$, and $q' \approx 2^{60}$ be two prime numbers and $Q = q \cdot q'$, and let R be the cyclotomic field of index 2^{13} (i.e. the ring modulo $\Phi(x) = X^{2^{12}} + 1$). Also let $n'_0 = 2$, $n'_1 = n'_2 = 3$, and let $m'_1 = n'_0 \cdot \lfloor \log_4 Q \rfloor = 2 \cdot 51 = 102$, and $m'_2 = n'_1 \cdot \lfloor \log_{2^{51}} Q \rfloor = 3 \cdot 2 = 6$. We use the gadget matrices $G_1 = (1, 4, 16, \ldots 4^{\frac{m'_1}{2}-1}) \otimes I_2 \in R_Q^{2\times 102}$ and $G_2 = (1, 2^{51}) \otimes I_3 \in R_Q^{3\times 6}$ for decryption (and $\hat{G} = (1, 4, 16, \ldots 4^{\frac{m'_1}{2}-1}) \otimes I_3 \in R_Q^{3\times 153}$ for the key-switching gadgets).

The size of the database is $N = N_1 \times N_2 \times \cdots \times N_D$ with $N_1 = 256$ and $N_2 = \ldots = N_D = 2$. Below we consider the example of $N = 2^{20}$ (so $D = 13$), but these parameters can support much larger databases (easily $D = 100$ or more).

Pre-Processing the Database. The server arranges the N-entry database into a hypercube of dimensions $N_1 \times N_2 \times \cdots \times N_t$, breaks every entry $DB(i_1, i_2, \ldots, i_D)$ into blocks of size $42 \cdot 2^{12} \cdot 4$ bits (about 84KB). Each block is interpreted as a 2-by-2 matrix M over the ring R_q, the server computes the 2-by-3 matrix $M' = M \times H \bmod q$, and encodes each element of M' in CRT representation in R_Q. (We remark that the entries of M' are in the range $[\pm q/2]$, but they encoded modulo the larger $Q = q \cdot q'$.)

Key Generation. The client chooses a secret key $S = (S'|I) \in R^{2\times 3}$, and also generates a key-switching gadgets for the automorphisms τ_{2^j} for $j = 0, 1, \ldots, 7$. The key-switching matrix for τ_{2^j} is 3-by-153 a matrix W_{2^j} over R_Q such that

$$S \times W_{2^j} = S_{2^j} \times \hat{G} + 2^{8-j} \cdot \hat{E}_{2^j} \pmod{Q}$$

where $S_{2^j} = \tau_{2^j}(S)$, and the entries of the noise matrices $\hat{E}_{2^j}$ are chosen from a distribution of variance 8 (which is good enough to avoid the Arora-Ge attacks [AG11]). Observe that the noise matrix in the gadget for τ_{2^j} is multiplied by 2^{8-j}, this keeps those matrices much smaller than Q but ensures that they are divisible by some powers of two. We will need it to implement the homomorphic division-by-two operation.

The client sends these eight key-switching gadgets to the server. They take up significant bandwidth, specifically a total of $8 \cdot 3 \cdot 153 \cdot 2^{12} \cdot 102$ bits, or about 183MB. However they are independent of the queries, and can be sent off-line once and used for many queries. (It is also likely that the key-switching technique from [GHS12] can be used to make these gadgets much smaller.)

Encrypting the Index i. The client expresses the target index $i \in [N]$ in a mixed-radix representation $(i_1, i_2, \ldots, i_D)$ with $i_1 \in [256]$ and $i_2, \ldots, i_D \in \{0, 1\}$.

- To encrypt i_1 the client encrypts the element $\sigma = X^{i_1} \in R$ by a ciphertext matrix $C_1 \in R_Q^{n'_1 \times m'_1}$ such that $SC_1 = q' \cdot \sigma_1 G_1 + 256 \cdot E_1 \pmod{Q}$, for a low-norm noise matrix E_1.
  Note that the ring element σ is multiplied by q' and the noise matrix E_1 is multiplied by 256. This still leaves the noise much smaller than Q, and is needed for the division-by-2 operations.

– For all $j > 1$, the client encrypts the bit i_j in a GSW ciphertext C_j such that $SC_j = i_j \cdot SG_2 + E_j \pmod{Q}$.

As before, the entries of the noise matrices E_j above are chosen from a distribution of variance 8.

The client sends to the server the ciphertexts $C_1, \ldots, C_D$. They consists of $n'_1 \cdot m'_1 = 306$ ring elements for C_1 and $D \cdot n'_1 \cdot m'_2 = 12 \cdot 3 \cdot 6 = 216$ elements for all the other C_j's, for a total of 522 ring elements in R_Q, or roughly 26MB.

Processing the first index i_1. The server applies the query-expansion procedure from fig. 2 homomorphically to the ciphertext C_1, resulting in $N_1 = 256$ ciphertexts $C_{1,j}$ that encrypt the bits of the i_1'st unit vector $(b_1, \ldots, b_{N_1})$ via the invariant $SC_{1,j} = q' \cdot b_j \cdot G_1 + E_{i,j} \pmod{Q}$.

The homomorphic implementation of additions and multiply-by-scalar are obvious, and homomorphic automorphism was described earlier in this section. To implement the division-by-two, the server just multiplies the ciphertext by the integer $2^{-1} \bmod Q = (Q+1)/2$. In Lemma 4 we prove that whenever we apply that operation, we have that $SC \bmod Q$ is even, hence multiplying by $2^{-1} \bmod Q$ indeed divides it by two. (As a side benefit, we also get that the noise is halved by this operations.)

The server next "shrinks" the dimension of the $C_{1,j}$'s, computing $C'_{1,j} = C_{1,j} G_1^{-1}(I_2) \in R_Q^{3\times 2}$, and it is easy to verify that we have $SC'_{1,j} = q' b_j I_2 + E'_{1,j}$, where $\|E'_{1,j}\| \le \|E_{1,j}\|$ (since each column of $G_1^{-1}(I_2)$ is all-0 except a single 1 entry). The server encodes each entry of $C'_{1,j}$ in CRT representation modulo Q.

Keeping everything in CRT representation, the server then multiplies each $C'_{1,j}$ by the pre-processed database row j: For each matrix $M' = MH \bmod q$ in the j'th row, the server computes $C = C'_{1,j} \times M' \bmod Q$. Finally, the server adds all the resulting encrypted rows. Recalling that all the $C'_{1,j}$'s encrypt zeros except C'_{1,i_1} that encrypts q', we get that the resulting $(D-1)$-dimensional hypercube encrypts the i_1'st row of the original database, times q' (with every matrix multiplied on the right by H modulo q).

Processing the other indexes $i_2, \ldots, i_D$. The server next continues to fold the size-2 dimensions one after the other. For each size-2 dimension it multiplies C_j (encrypting i_j) by the encrypted second column, and the ciphertext $G - C_j$ (encrypting $1 - i_j$) by the encrypted first column. Adding the two we get an encryption of the i_j'th column (multiplied by q' and by H on the right). This continues until the server is left with just a single encrypted database entry, consisting of some number of compressed ciphertexts modulo Q.

Modulus Switching and Reply. Finally the server performs modulus switching, replacing each ciphertext C by $C' = \lceil C/q' \rfloor \in R_q^{3\times 3}$, and sending the resulting ciphertexts to the client for decryption. The ciphertext C before modulus switching satisfied $SC = q'MH + E \pmod{q'q}$. Denoting the rounding error by Ξ, the new ciphertext has

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

Since the key S was chosen from the error distribution and $\|\Xi\|_\infty \le 1/2$, then the added noise is small and the result is a valid ciphertext. (See more details below.) The server then sends the ciphertexts back to the client to be decrypted.

A.4 Analysis

Correctness. The only step where correctness is non-obvious is the query expansion homomorphic procedure. The following lemma establishes that it indeed computes the right thing.

Lemma 4. *For each iteration $j = 0, 1, \ldots, 7$ of query expansion, at the end of the j'th iteration and for all $k \in [2^{j+1}]$, the entry $\mathsf{list}[k]$ contains a ciphertext C_k satisfying*

$$S \times C_k \bmod Q = \begin{cases} q' \cdot X^{i_1 - k} + 2^{7-j}E & \text{if } i_1 = k \pmod{2^{j+1}} \\ 2^{7-j}E & \text{otherwise,} \end{cases}$$

for a low-norm matrix E. Specifically, the noise has the form

$$2^{7-j}E = 2^{7-j}\left(\sum_{i=0}^{2^{j+1}-1} E_i + \sum_{i=0}^{2^{j+1}-2} E'_i \hat{G}^{-1}(\mathsf{something}_i)\right)$$

where the E_i's and E'_i's are fresh noise matrices with entries of variance 8.

Proof. The above clearly holds at the beginning of the protocol before iteration $j = 0$ (substituting $j = -1$), this is how the ciphertext C_1 was encrypted. Assume then that it holds after the $j-1$'st iteration and we prove for j.

We begin by analyzing with the noise term. Consider some $k \in [2^j]$, the j'th iteration begins with $\mathsf{list}[k]$ containing a ciphertext C such that $SC = \mathsf{somePlaintext} + 2^{8-j}\big(\sum_{i=0}^{2^j-1} E_i + \sum_{i=0}^{2^j-2} E'_i \hat{G}^{-1}(\mathsf{something}_i)\big)$. For the cyclotomic ring modulo $X^M + 1$, multiplying by a monomial X^r (in line 6) and applying automorphisms (in lines 7-8) induce permutations of the polynomial coefficients, which do not change the noise magnitude or its variant. Also, the key-switching at the end of the homomorphic automorphisms in this iteration only adds a noise term of the form $2^{8-j}E_{2^j}$. Hence the noise terms in the ciphertexts that are stored in entries k and $k + 2^j$ in lines 7-8 have the form

$$\begin{aligned}
&\frac{1}{2}(2^{8-j}E + 2^{8-j}E' + 2^{8-j}E_{2^j}\hat{G}^{-1}(\mathsf{something})) \\
&= 2^{7-j}\Bigg(\big(\sum_{i=0}^{2^j-1} E_i + \sum_{i=0}^{2^j-2} E'_i \hat{G}^{-1}(\mathsf{something}_i)\big) \\
&\qquad + \big(\sum_{i=2^j}^{2^{j+1}-1} E_i + \sum_{i=2^j-1}^{2^{j+1}-2} E'_i \hat{G}^{-1}(\mathsf{something}_i) + E_{2^j}\hat{G}^{-1}(\mathsf{something})\big)\Bigg) \\
&= 2^{7-j}\Bigg(\sum_{i=0}^{2^{j+1}-1} E_i + \sum_{i=0}^{2^{j+1}-2} E'_i \hat{G}^{-1}(\mathsf{something}_i)\Bigg).
\end{aligned}$$

For the "signal" (i.e., the encrypted value), we use the argument from the proof of [ACLS18, Thm. 1]. Let $\chi_{2^j}(i_1 = k)$ be the indicator function which is 1 if $i_1 = k \pmod{2^j}$ and 0 otherwise. We begin the iteration with an encryption of $\chi_{2^j}(i_1 = k) \cdot q' \cdot X^{i_1 - k}$.

If $i_1 \neq k \pmod{2^j}$ then this is an encryption of zero, and it remains so when multiplying by X^r (in line 6) and applying automorphisms (in lines 7-8), and therefore also when performing the

additions. In that case we have an encryption of an even number (zero) with even noise (per the analysis above), so multiplying by $2^{-1} \bmod Q$ yields a valid encryption of zero.

For the case $i_1 = k \pmod{2^j}$, denoting $r = (i_1 - k)/2^j$ (which is an integer). We begin with an encryption of $q' \cdot X^{i_1-k} = q' \cdot X^{r2^j}$, and in line 6 we multiply it by the scalar X^{-2^j}, getting an encryption of $q' \cdot X^{(r-1)2^j}$.

Applying automorphism-by-t to $q' \cdot X^{r2^j}$ (in line 7), we get an encryption of $q' \cdot X^{r2^j \cdot t}$. With $t = M/2^j + 1$ we have $r2^j \cdot t = rM + r2^j$, and since $X^M = -1$ in the ring then $X^{r2^j \cdot t} = (-1)^r \cdot X^{r2^j} = (-1)^r \cdot X^{i_1-k}$. If r is even then the encrypted value after the addition in line 7 is $2q'X^{i_1-k}$. Again we have an encryption of an even value with even noise term, so multiplying by $2^{-1} \bmod Q$ yields a valid encryption of $q'X^{i_1-k}$. This is exactly what we need, since r being even means that $i_1 = k \pmod{2^{j+1}}$. If r is odd then the same argument as above applies, except that instead of $2q'$ we have 0, which is what we need because in this case $i_1 \neq k \pmod{2^{j+1}}$.

Applying the automorphism in line 8, we get an encryption of $q' \cdot X^{(r-1)2^j \cdot (M/2^j+1)} = q' \cdot (-1)^{r-1} \cdot X^{(r-1)2^j}$. If r is even then this is an encryption of $-q' \cdot X^{(r-1)2^j}$, so we store an encryption of zero in $\mathsf{list}[k + 2^j]$, which is what we needed since $i_1 \neq k + 2^j \pmod{2^{j+1}}$. If r is odd then we get an encryption of $(q'+q')/2 \cdot X^{(r-1)2^j} = q' \cdot X^{i_1-(k+2^j)}$, again as needed when $i_1 = k + 2^j \pmod{2^{j+1}}$. □

Noise analysis. We begin by bounding the variance of the noise term, under the heuristic assumption that the different terms behave as if they were independent of each other.

Processing the first index i_1. Recall that by Lemma 4, the noise terms at the end of query expansion had the form $\sum_{j=0}^{256} E_j + \sum_{j=0}^{255} E'_j \hat{G}^{-1}(\mathsf{something}_j)$. The entries of the E's and E''s have variance 8, and those of $\hat{G}^{-1}(\cdot)$ are in the range $[\pm 2]$ (because we have $m'_1 = n'_1 \log_4(Q)$). Since each entry in $E'_j \hat{G}^{-1}(\mathsf{something})$ is a sum of $2^{12} \cdot 153$ terms, each is a product of a variance-8 variable with a number in $[\pm 2]$, then (under the independence heuristic) the variance of each entry is below $2^{12} \cdot 153 \cdot 4 \cdot 8 = 153 \cdot 2^{17}$. It follows that each noise term after query expansion has variance bounded by $256 \cdot 8 + 255 \cdot 153 \cdot 2^{17} < 305 \cdot 2^{24}$.

Shrinking the ciphertexts via multiplication by $G^{-1}(I_2)$ does not increase the variance, and multiplying by the matrices $MH \bmod q$ (of entries in $[\pm 2^{41}]$) increases the variance by a factor of at most $2^{2 \cdot 41} \cdot 3 \cdot 2^{12} = 3 \cdot 2^{94}$. Hence each encrypted row has noise terms of variance at most $305 \cdot 2^{24} \cdot 3 \cdot 2^{94} = 915 \cdot 2^{118}$. Since we are adding 256 rows, then the variance of each noise coordinate after processing the first encrypted index i_1 is bounded by $915 \cdot 2^{126}$.

Processing the other indexes i_j. As we continue to fold more dimensions, we multiply the encrypted unary vectors for those dimensions (which are GSW ciphertexts) by the results of the previous dimension (which are compressed ciphertexts) using Equation (5), using the 3-by-6 gadget matrix 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. Encrypting these GSW ciphertext therefore adds a noise term of the form $E \times G_2^{-1}(\mathsf{something})$. Moreover, only one of the two terms in each dimension has an encrypted bit $\sigma = 1$ while the other have $\sigma = 0$. Hence the term $\sigma \cdot \mathsf{previousNoise}$ appears *only once* in the resulting noise term after folding the j'th dimension. Therefore folding each small dimension $j \geq 2$ just adds two noise terms of the form $E \times G^{-1}(\mathsf{something})$ to the noise from the previous dimension.

Since G_2 has $m'_2 = n'_1 \log_{2^{51}}(Q)$, then each entry in G_2^{-1} is in the interval $[\pm 2^{50}]$, and multiplying by G_2 increases the variance by a factor of less than $(2^{50})^2 \cdot m'_2 \cdot 2^{12} = 3 \cdot 2^{113}$. With $2(D-1) = 24$ of these terms, the added variance is bounded by $24 \cdot 8 \cdot 3 \cdot 2^{113} = 9 \cdot 2^{119}$.

Modulus switching. The analysis so far implies that prior to the modulus switching operation, the noise variance is bounded by $915 \cdot 2^{126} + 9 \cdot 2^{119} < 916 \cdot 2^{126}$. Scaling down by a factor $q' \approx 2^{60}$ decreases the variance by a factor of roughly 2^{120}, resulting in scaled noise of variance bounded by $916 \cdot 2^6$. 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 the noise variance of the final ciphertext after modulus switching is bounded below $916 \cdot 2^6 + 3 \cdot 2^{15} = 156928 \approx 2^{17.26}$.

Bounding the error. Since each noise coefficient consists of the weighted sum of many noise terms with zero mean and similar variance, it makes sense to treat it as a Normal random variable. Recalling that we use the nearly square gadget matrix H with $p = \sqrt[3]{q} \approx 2^{42/3} = 2^{14}$, we can handle noise as large as $(p-1)/2 = 2^{13}$. To get a decryption error, at least one of the noise coefficients will need to be larger than 2^{13} in magnitude, namely it must be more than 20 standard deviations away from its mean. The probability for any one Normal variables to be so far from its mean is below $\mathsf{erfc}(20/\sqrt{2}) < 2^{-293}$, with $9 \cdot 2^{12}$ coefficients the probability of error is below 2^{-277}.

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_1 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 Regev ciphertexts by the plaintext matrices encoding the database, while folding the first hypercube dimension. These multiplications have the form $C' := C \times MH \bmod q'q$ with C a ciphertext of dimension $n_1' \times n_0'$ and $M'H$ a redundant plaintext matrix of dimension $n_0' \times n_2'$, with $n_0' = 2$ and $n_1' = n_2' = 3$ (all over the ring R_Q). Using the naïve matrix-multiplication algorithm, we need $3^2 \cdot 2 = 18$ ring multiplications for each of these matrix multiplications, modulo the double-sized modulus $Q = q' \cdot q$. 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 $2 \cdot 18 \cdot 2^{12} = 9 \cdot 2^{14}$ modular multiplications.

For this work, we process a single 2-by-2 plaintext matrix over R_q, containing of $4 \cdot 2^{12} \cdot 42$ bits, namely $42 \cdot 2^{11}$ bytes. Hence the amortized work per byte is $\frac{9 \cdot 2^{14}}{42 \cdot 2^{11}} = \frac{9 \cdot 8}{42} \approx 1.72$ modular multiplication per database byte. (We can even reduce this number slightly by using better methods of matrix multiplication.) 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.

Less than two 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 [SR10,Cry09]. 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.