Efficient Multi-Query CPIR from Ring-LWE*

Helger Lipmaa[1,2]

University of Tartu, Estonia

Abstract. We propose an (n, m)-computationally-private information retrieval (CPIR) protocol with rate $1 - o(1)$ and highly nontrivial (sublinear and data-dependent) server's computational complexity. For this, we note that an (n, m)-CPIR protocol is equivalent to a secure function evaluation protocol that evaluates a secret function f on m different inputs. Thus, we first design an efficient multi-level circuit for f, and then use the recent (ring-)LWE based fully-homomorphic encryption scheme by Brakerski, Gentry and Vaikuntanathan [BGV11] to evaluate the circuit in a private manner. Apart from the final result itself, several of our techniques may be of independent interest. This includes the construction of the circuit for f and the definition and construction of computational batch codes.

Keywords. Circuit complexity, compressed constant-weight codes, computational batch codes, CPIR, parallel computation, ring-LWE.

1 Introduction

In many privacy-preserving data mining applications, the client needs to retrieve in parallel some (possibly many) items out of the server's database, without the server realizing which elements were fetched. A typical example consists of a company executive accessing the salary information of the employees to be fired, or a medical doctor accessing biographic information of the patients who are scheduled for certain tests the same day. In many such cases, one needs to retrieve a large number of database elements at once. The cryptographic tool that implements this functionality is called (multi-query) computationally-private information retrieval (CPIR, [CGKS95,KO97]).

More technically, in an m-out-of-n computationally-private information retrieval ((n, m)-CPIR) protocol, the client obtains m different elements of his choice from the server's n-element database of ℓ-bit strings. The server obtains no information about which m elements were retrieved. There are $\binom{n}{m}$ different choices for the client to select m different elements out of n. In a non-private version of (n, m)-CPIR, the client just sends the number of the corresponding choice $0 \leq y < \binom{n}{m}$ to the server, who replies with m ℓ-bit database elements. The total communication of such a protocol is $\lceil \log_2 \binom{n}{m} \rceil + m\ell$ bits. As shown in [GKL10], $\lceil \log_2 \binom{n}{m} \rceil + m\ell$ is also a communication complexity lower bound even for *multiround* and *non-private* information retrieval protocols.

In the private case, one is interested in the communication rate (the amount of useful information, $\lceil \log_2 \binom{n}{m} \rceil + m\ell$, divided by the communication complexity of the (n, m)-CPIR protocol) and the server's computational complexity. (The client's computational complexity is much smaller than the one of the server in almost all well-known CPIR protocols, and for this reason we will not study it.) The only prior-art multi-query CPIR with constant rate was recently proposed in [GKL10], however their protocol has server's computational complexity of $\Omega(mn\ell/\kappa)$ public-key operations, see Tbl. 1. Clearly, this is computationally too inefficient for large values of m. (Here and in what follows, κ is a security parameter that corresponds to the ciphertext length, and λ is a security parameter that corresponds to the exponential security. E.g., $\lambda = 80$.) On the other hand, the only prior-art sublinear-communication $(n, 1)$-CPIR that also achieved server's sublinear-in-n computational complexity of $O(n\ell/\log(n\ell))$ public-key operations was proposed in [Lip09]. (This given estimation is an upperbound, the amount of actual computation depends on the BDD-complexity of the server's database.) Its m-times parallel repetition has communication complexity $\Theta(\kappa m \cdot \log^2 n + m\ell)$, which is far from optimal. See Sect. 2 for related work.

In [IKOS04], the authors showed how to reduce the computation of (n, m)-CPIR protocols by using batch codes. Recall that in an $[n, N, m, M, T]_\ell$ batch code, a database $\boldsymbol{f}$ of n ℓ-bit strings is divided into M bins where each bin contains N/M ℓ-bit strings. If a client wants to obtain m elements of the original database, he will need to query (no more than) T elements from each of the M bins. A batch code guarantees that based on the answers to the resulting $\leq M \cdot T$ queries, the client is able to efficiently reconstruct the m elements he was originally interested in.

* Draft, August 30, 2011

Protocol	Upperbound on server's computation bit-operations	Communication in bits	Assumption
Prior art			
[GKL10]	$\Omega(mn\ell/\kappa)\cdot\Omega(\lambda^6)$	$\le 100\cdot(\log_2\binom{n}{m}+m\ell+\kappa)$	DSA + PRG
[Lip09] + ex	$O(n\ell\log n/\log(n\ell\log n/m))\cdot\Omega(\lambda^6)$	$\Theta(\kappa m\cdot\log^2(n\log n/m)+m\ell)$	DCRA
[Lip09] + ss	$O(n\ell/\log(n\ell/m^2))\cdot\Omega(\lambda^6)$	$\Theta(\kappa m^2\cdot\log^2(n/m^2)+m^2\ell)$	DCRA
Old CPIR protocols with the new computational batch code			
[Lip09] + cb	$O(n\ell/\log(Tn\ell/m))\cdot\Omega(\lambda^6)$	$\Theta(\kappa m\cdot\log^2(Tn/m)/T+m\ell/T)$	DCRA + PRG
New CPIR: versions with different rate			
$1/\kappa$	$\Theta\left(\frac{Tn\ell}{((Tn/m)^{\log_2\log_2(Tn/m)}\log_2(Tn\ell/m))}\right)\cdot\tilde{O}(\lambda^2)$	$((1+o(1))\cdot\log_2\binom{n}{m}+2m(\ell+1))\kappa$	RLWE + PRG
$\frac{1}{\operatorname{polylog}\lambda}$	$\Theta\left(\frac{Tn\ell}{((Tn/m)^{\log_2\log_2(Tn/m)}\log_2(Tn\ell/m))}+\mathsf{T}_{\mathsf{sym}}\left(\log_2\binom{n}{m}\right)\right)\cdot\tilde{O}(\lambda^2)$	$\lceil\log_2\binom{n}{m}\rceil+(1+o(1))m\ell$	RLWE + PRG

Table 1. Efficiency comparison of different (n,m)-CPIR protocols. Here, "ex"/"ss" is the expander/subset code from [IKOS04], "cb" is the computational batch code from the current paper. We omit the communication cost of sending the public key. In all protocols but [GKL10], the server's computational complexity depends on the database, and the value given here corresponds to the worst-case computational complexity. DCRA denotes the composite residuosity assumption and DSA denotes the decision subgroup assumption from [GKL10]. See the text for the definition of T and $\mathsf{T}_{\mathsf{sym}}$

Given an $[n,N,m,M,T]_\ell$ batch code, the client and the server can replace an instance of a (n,m)-CPIR protocol with M parallel instances of $(N/M,T)$-CPIR protocols. For example, one can run Lipmaa's $(N/M,1)$-CPIR protocol from [Lip09] M times in parallel to obtain a batch-coded (n,m)-CPIR protocol. The resulting protocol has server's computational complexity $\Theta(M\cdot((N/M)\ell/\log(N/M\cdot\ell)))=\Theta(N\ell/\log(N\ell/M))$ public-key operations and communication complexity $\Theta(\kappa M\cdot\log^2(N/M)+M\ell)$. The best batch code in [IKOS04] (namely, the expander code) with $M=O(m)$ has $N=O(n\log n)$ and $T=1$. On the other hand, the subset code from [IKOS04] has $N=O(n)$ but $M=\Omega(m^2)$ and $T=1$, see Sect. 5. Lipmaa's CPIR protocol from [Lip09] batched with either of those two codes is still far from optimal. See rows 2 and 3 in Table 1.

Our Contributions. We propose a new (n,m)-CPIR protocol with rate $1-o(1)$ that at the same time achieves highly nontrivial (sublinear and data-dependent) server's computational complexity. The new CPIR has two versions. The first version has asymptotically the best server's computational complexity among any known multi-query CPIR protocols, and rate $1/\kappa$ where $\kappa=\tilde{O}(\lambda)$. The second version has asymptotically optimal rate $1-o(1)$ but a somewhat higher server's computational complexity. Our constructions constitute of the next three principal steps, each of which by itself contains some new results.

First. Following [Lip09], we interpret the server's database $\boldsymbol{f}=(f_1,\ldots,f_n)$ as a function f with $f(i)=f_i$. Thus we think of an (n,m)-CPIR protocol as a secure function evaluation protocol that computes f in parallel on m different inputs. We design a circuit for computing the sequence $(f(x_1),\ldots,f(x_m))$, where the input $\boldsymbol{x}=(x_1,\ldots,x_m)$, with $0\le x_i<x_{i+1}<n$, is represented by an encoded integer $0\le y<\binom{n}{m}$. By evaluating this circuit, the server clearly evaluates nonprivate version of information retrieval. Moreover, the resulting nonprivate protocol has optimal communication.

Unfortunately, for most databases $\boldsymbol{f}$ it is a priori unclear how to design such an optimal circuit, or even how large it is. As an example, by a classical result of Lupanov [Lup58], any function $F:\{0,1\}^n\to\{0,1\}^\ell$ can be computed by a circuit of size $(1+o(1))2^n\ell/\log_2(2^n\ell)$. Since in our case, $f:\left\{0,\ldots,\binom{n}{m}-1\right\}\to\{0,1\}^{m\ell}$, one can design a circuit with size $(1+o(1))\binom{n}{m}\cdot m\ell/\log_2(\binom{n}{m}\cdot m\ell)$ that computes f. Unfortunately, this circuit has a superpolynomial-in-n size if $m=\Omega(\log n)$.

Thus, to construct an efficient protocol based on fully-homomorphic encryption, one has to first design an efficient circuit for the problem at hand, and the later task is often non-trivial by itself. In particular, the current paper makes some contributions to the field of circuit complexity.

More precisely, our circuit consists of two levels. In the first level, an $\approx\log_2\binom{n}{m}$-bit input y is decoded to a codeword $(x_0,\ldots,x_{m-1})$, $0\le x_i<x_{i+1}<m$ in what we call the compressed constant-weight code by using recent algorithms by Tian, Vaishampayan and Sloane [TVS09,SV09] that are described in Sect. 3. The second level circuit outputs, given input $(x_0,\ldots,x_{m-1})$, the tuple $(f(x_0),\ldots,f(x_{m-1}))$. To do the latter efficiently, we generalize a little-known result of Uhlig [Uhl74,Uhl92] to the case of non-Boolean functions f. That is, we prove that if $m<n^{1/\log_2\log_2 n}$, then the size of the second level circuit is upper bounded by $(1+o(1))n\ell/\log_2(n\ell)$, i.e., it does not depend on m at all. This is a contribution of independent interest. Clearly, if m is larger, one can just

compute $\lceil m/n^{1/\log_2\log_2 n}\rceil$ circuits of size $(1+o(1))n\ell/\log_2(n\ell)$ in parallel. See Sect. 4. If $m = O(\sqrt{n/\log n})$, then the first level circuit is small. The new generalization of Uhlig's result decreases the size of the second-level circuit (and thus also of the whole circuit) by a factor of $n^{1/\log_2\log_2 n}$.

Second. For the first-level circuit to be efficient, we needed $m = O(\sqrt{n/\log n})$. (We call a CPIR protocol that satisfies this condition *restricted*.) Even then, for large values of m the resulting circuit is too large. To this end, we define computational $[n, N, m, M, T]_\ell$ batch codes that guarantee $(1-2^{-\lambda})$-completeness under a computational assumption, while batch codes of [IKOS04] have unconditionally guaranteed perfect completeness. (This can be compared with the work on computational error-correcting codes [MPSW05].) Due to the relaxed completeness conditions, our new computational $[n, N = n, m, M = \lceil m/T\rceil, 2T]_\ell$ batch code is significantly more efficient than any of the batch codes proposed in [IKOS04].

While the new computational batch code is inspired by an implicit construction of [GKL10], we have provided a full definitional framework, and have made it more efficient by using a significantly more precise argument to bound the completeness error: namely, according to the analysis of [GKL10], $T = \Omega(\lambda)$ while in our case $T = \Theta(\sqrt{m\lambda/(n(n-m))}) = O(\sqrt{\lambda})$ and in practice one can assume $T \leq 10$.[1]

Given the new computational batch code, we need to design $\lceil m/T\rceil$ considerably smaller circuits for $(\lfloor Tn/m\rfloor, 2T)$-CPIR protocols. As mentioned above, in the batch codes of [IKOS04] either $N = \Omega(n\log n)$ or $M = \Omega(m^2)$ while we achieve simultaneously $N = n$ and $M = \lceil m/T\rceil$.

Third. On top of the previous constructs, we use the recent ring-LWE [LPR10] based fully-homomorphic encryption scheme by Brakerski, Gentry and Vaikuntanathan [BGV11] BGV to homomorphically evaluate the $\lceil m/T\rceil$ circuits; this clearly implements the (n, m)-CPIR protocol.[2] The concrete FHE scheme was chosen because of its relative efficiency. We also need its details later to decrease the rate of the CPIR protocol. In the case of an optimal underlying circuit, the just described CPIR protocol has optimal server's computational complexity (times the time to evaluate the fully-homomorphic encryption scheme once, which is $\tilde{O}(\lambda^2)$ bit-operations in the case of the BGV cryptosystem) and rate $1/\kappa$. For an arbitrary database, by using the results desribed above, we obtain the upper bound of row 5 in Tbl. 1.

To improve on this result, we use two separate tricks to compress the client's message and the server's message. More precisely, we improve the rate to $1 - o(1)$. However, the improved rate protocol has some additional computational cost, see row 6 of Tbl. 1 where $\mathsf{T}_{\mathsf{sym}}(x)$ is the circuit-complexity of some CPA-secure symmetric cryptosystem on x-bit input.

Finally, as estimated in [BGV11], one public-key operation of the BGV cryptosystem takes $\tilde{O}(\lambda^2)$ bit-operations. At the same time, all previous (not based on fully-homomorphic encryption) efficient CPIR protocols [Lip05,GR05,Lip09,GKL10] use public-key operations that have bit-complexity $\Omega(\lambda^6)$.[3] Currently, this is offset by the complexity of key-generation and the length of public keys of the existing fully-homomorphic encryption schemes, including [BGV11]. However, we expect that due to the pace of research in the area of fully-homomorphic encryption, more efficient FHE schemes will be constructed within few years. We hope that non-trivial protocol papers like the current one will give additional motivation for such research. Moreover, in practice the main bottleneck that does not allow one to implement CPIR (and oblivious transfer) protocols is their computational complexity.

Efficient $(n, 1)$-CPIR Protocols. Let $L(f)$ be the size of the optimal circuit that "computes" f. As a simple corollary, if $m = 1$ then from the previously described rate-$1/\kappa$ (n, m)-CPIR protocol we obtain an $(n, 1)$-CPIR protocol with communication complexity $\Theta(\kappa\cdot\log n + \kappa\ell)$ and the server's computational complexity of $\Theta(L(f))\cdot\tilde{O}(\lambda^2)$ bit-operations. For this, we note that in the previous construction, if $m = 1$ then we do not need the part of encoding/decoding the compressed constant-weight codes. One can further improve the rate by using techniques from Sect. 6. This result is interesting by itself since it also provides an efficient protocol for secure function evaluation.

The only previous $(n, 1)$-CPIR protocol with polylogarithmic communication and sublinear computation was proposed recently by Lipmaa [Lip09]. Lipmaa's $(n, 1)$-CPIR protocol has communication complexity $\Theta(\kappa\cdot\log^2 n + \ell)$ and computation complexity of $\Theta(L_{\mathsf{bp}}(f))\cdot\Omega(\lambda^6)$ online bit-operations, where $L_{\mathsf{bp}}(f)$ is

[1] Most probably, one can use this result to make the (n, m)-CPIR protocol of [GKL10] more efficient, but investigating this is out of scope of the current paper.

[2] We can also use the LWE-based variation of the cryptosystem from [BGV11]. The resulting CPIR protocol will be somewhat less efficient, but it will be based on a more standard underlying assumption (LWE instead of ring-LWE).

[3] The $(n, 1)$-CPIR protocol of [BV11] is based on public-key operations that has bit-complexity $\Omega(\lambda^4)$ according to [BGV11]. Obviously, one can implement their CPIR protocol on top of the BGV cryptosystem and achieve $\tilde{O}(\lambda^2)$ bit-operations per public-key operation.

the size of the optimal branching program (also known as a binary decision diagram) that "computes" f. Note that often, $L_{\mathsf{bp}}(f) > L(f)$, and thus the new $(n,1)$-CPIR protocol is asymptotically both more communication-efficient and computation-efficient than Lipmaa's $(n,1)$-CPIR protocol from [Lip09]. (Moreover, in the protocol of [Lip09], the online public-key operations are applied on long inputs which makes the public-key operations and thus also the whole protocol even less computationally efficient.) Note that Gentry [Gen09a] proposed another $(n,1)$-CPIR protocol, based on an FHE scheme, with communication complexity $\Theta(\kappa \cdot \log n + \kappa\ell)$ and significantly larger computational complexity of $\Theta(n \cdot \log n)$ (fully-homomorphic) public-key operations. A similar $(n,1)$-CPIR protocol but with the linear computational complexity is imminent from [Lip08] and was recently constructed in [BV11].

2 Preliminaries

Notation. Vectors will be denoted in bold like in $\boldsymbol{f} = (f_0, \ldots, f_{n-1})$. We make use of the Landau notations $O(\cdot)$, $\Omega(\cdot)$, $\tilde{O}(\cdot)$, etc. A positive function $g(n)$ is negligible if $g(n) = n^{-\omega(1)}$. A positive function $g(n)$ is overwhelming if $1 - g(n)$ is negligible. If X is a random variable, then $E[X]$ denotes its mean and $\Pr[X = x]$ denotes the probability of the event $X = x$. κ and λ are security parameters, κ is usually equal to the ciphertext length, while λ corresponds to the exponential security (every attack takes time at least 2^λ); λ is also the security parameter for the used symmetric encryption scheme. All logarithms have basis 2. Note that $\lceil \log_2 \binom{n}{m} \rceil \approx \lceil n \cdot H_2(m/n) \rceil = \lceil n \log_2 n - m \log_2 m - (n-m)\log_2(n-m) \rceil$ bits, where $H_2(p) := -p \cdot \log_2 p - (1-p) \cdot \log_2(1-p)$ is the binary entropy function.

Ring-LWE. For security parameter λ, let $f(x) = x^d + 1$, where $d = d(\lambda)$ is a power of 2. Let $q = q(\lambda) \geq 2$ be an integer. Let $R = \mathbb{Z}[x]/(f(x))$ and $R_q = R/qR$. In particular, $|R_q| = q^d$. For $a \in R$, let $[\![a]\!]_q$ denote $a \mod q$ with coefficients reduced into the range $(-q/2, q/2]$.

Let $\chi = \chi(\lambda)$ be a distribution over R. The $\mathsf{RLWE}_{d,q,\chi}$ problem is to distinguish between the following two distributions. In the first distribution, one samples (a_i, b_i) uniformly from R_q^2. In the second distribution, one draws $s \leftarrow R_q$ uniformly and then samples $(a_i, b_i) \leftarrow R_q^2$ by sampling $a_i \leftarrow R_q$ uniformly, $e_i \leftarrow \chi$, and setting $b_i \leftarrow a_i \cdot s + e_i$. The $\mathsf{RLWE}_{d,q,\chi}$ assumption [LPR10] is that the $\mathsf{RLWE}_{d,q,\chi}$ problem is infeasible.

In [LPR10], the authors proved that for any d that is a power of 2, ring $R = \mathbb{Z}[x]/(x^d + 1)$, prime integer $q = q(d) \equiv 1 \pmod{d}$, and $B = \omega(\sqrt{d \log d})$, there is an efficiently sampleable distribution χ that outputs elements of R of length at most B with overwhelming probability, such that if there exists an efficient algorithm that solves $\mathsf{RLWE}_{d,q,\chi}$, then there is an efficient quantum algorithm for solving $d^{\omega(1)} \cdot (q/B)$-approximate worst-case shortest vector problem for ideal lattices over R. Typically, χ is a discrete Gaussian distribution, where the vectors have length $\mathrm{poly}(d)$ with overwhelming probability.

Fully-Homomorphic Encryption Schemes. Let $\mathsf{PKC} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be a public-key cryptosystem. Fix the length ℓ of the elements of the plaintext space Msg. Gen is a randomized key generation algorithm such that $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\kappa)$; Enc is a randomized encryption algorithm such that for any $M \in \mathsf{Msg}$, $C \leftarrow \mathsf{Enc}_{\mathsf{pk}}(M)$; Dec is a decryption algorithm such that for any $C \in \mathsf{Cfr}$, where Cfr is ciphertext space, $M \leftarrow \mathsf{Dec}_{\mathsf{sk}}(C)$. We say that PKC is fully-homomorphic, if for any polynomial-size circuit $\mathcal{C}$, given ciphertexts $C_i = \mathsf{Enc}_{\mathsf{pk}}(M_i)$ of all k inputs to circuit $\mathcal{C}$, $\mathsf{Eval}_{\mathsf{pk}}(\mathcal{C}, C_1, \ldots, C_k)$ outputs a ciphertext C such that $\mathsf{Dec}_{\mathsf{sk}}(C) = \mathcal{C}(M_1, \ldots, M_k)$. The first (polynomial-time) fully-homomorphic encryption scheme was defined by Gentry [Gen09b].

We now formally define the CPA (*chosen plaintext attack*) game. First, the challenger chooses $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\kappa)$ and sends pk to the attacker. Second, the attacker selects some plaintexts M_0 and M_1 and sends them to the challenger. Third, the challenger chooses $b \leftarrow \{0,1\}$ and sends $\mathsf{Enc}_{\mathsf{pk}}(M_b)$ back to the attacker. Fourth, the attacker outputs a bit $b' \in \{0,1\}$. The attacker wins if $b' = b$. We say that a public-key cryptosystem is *CPA-secure* if no nonuniform probabilistic polynomial-time attacker wins with non-negligible probability.

The BGV cryptosystem. Based on techniques from [BV11], in a recent preprint [BGV11], Brakerski, Gentry and Vaikuntanathan proposed the up to now most efficient FHE scheme that can be instantiated either by the LWE or by the ring-LWE assumption. Since this BGV cryptosystem is yet not well known, we will next give a very brief overview of the more efficient ring-LWE based BGV scheme. See [BGV11] for justification of the parameters and a full description (the following description is definitely not complete).

Basic encryption scheme. In a setup algorithm, one chooses d to be a power of 2, $q \equiv 1 \pmod{d}$, $N = \lceil 3 \log_2 q \rceil$, $R = \mathbb{Z}[x]/(x^d + 1)$, and a χ such that the scheme achieves $\mathsf{RLWE}_{d,q,\chi}$ security of order $2^{-\lambda}$. In particular, assume that χ outputs an element of R of length at most $B_\chi = \omega(\sqrt{d \log d})$. To achieve 2^λ security against known lattice attacks, one must set $d = \Omega(\lambda \cdot \log(q/B))$. Let $params = (q, d, N, t, \chi)$.

The secret key is $s \leftarrow \chi$, a element of R_q. To generate a public key, generate uniform $\boldsymbol{a} \leftarrow R_q^N$ and $\boldsymbol{e} \leftarrow \chi^N$. Let $\boldsymbol{pk} = (\boldsymbol{\psi_1}, \boldsymbol{\psi_2}) = (\boldsymbol{a}, s\boldsymbol{a} + 2\boldsymbol{e}) \in R_q^{N\times 2}$. To encrypt $m \in R_2$, generate $\boldsymbol{r} \leftarrow R_2^N$ and set $\boldsymbol{c} = (c_1, c_2) \in R_q^2$, where $c_1 = m + \langle \boldsymbol{r}, \boldsymbol{\psi_2}\rangle = m + \sum_{i=1}^N r_i\psi_{2i} \in R_q$ and $c_2 = -\langle \boldsymbol{r}, \boldsymbol{\psi_1}\rangle = -\sum_{i=1}^N r_i\psi_{1i} \in R_q$. To decrypt (c_1, c_2), output $[\![[c_1 + c_2 s]_q]\!]_2$.

The noise of a fresh ciphertext is upperbound by $2dNB_\chi$, and this basic encryption scheme is correct with an overwhelming probability if that noise is less than $q/2$. Note that since $N = \Theta(\log q)$, $B_\chi = \Omega(\sqrt{d \log d})$ with $d = \Theta(\lambda \log q)$ then it suffices to take $q = \tilde{O}(\lambda^{1.5})$. The basic encryption scheme is secure under the $\mathsf{RLWE}_{d,q,\chi}$ assumption under the chosen parameters.

Adding/multiplying ciphertexts. Given two ciphertexts $\boldsymbol{c_1}, \boldsymbol{c_2} \in R^2$ (that encrypt m_1 and m_2 by using the same s, $\boldsymbol{a}$ and $\boldsymbol{e}$, and some different $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$, respectively), define $\mathsf{Add}(\boldsymbol{c_1}, \boldsymbol{c_2}) = \boldsymbol{c_1} + \boldsymbol{c_2} \in R^2$. Clearly, $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Add}(\boldsymbol{c_1}, \boldsymbol{c_2})) = \mathsf{Dec}_{\mathsf{sk}}(\boldsymbol{c_1}) + \mathsf{Dec}_{\mathsf{sk}}(\boldsymbol{c_2})$. The noise of $\mathsf{Add}(\boldsymbol{c_1}, \boldsymbol{c_2})$ is roughly equal to the sum of noises of $\boldsymbol{c_1}$ and $\boldsymbol{c_2}$.

Let $\mathsf{Mult}(\boldsymbol{c_1}, \boldsymbol{c_2}) = \boldsymbol{c'} = (c'_1, c'_2, c'_3) = (c_{11}c_{21}, c_{12}c_{21} + c_{11}c_{22}, c_{12}c_{22}) \in R^3$. One can "decrypt" $\boldsymbol{c'}$ by using secret key $\boldsymbol{s'} = (1, s, s^2)^\perp$ as follows: $[\![[\langle \boldsymbol{c'}, \boldsymbol{s'}\rangle]_q]\!]_2 = m_1 m_2$ if there resulting noise is less than $q/2$. The noise of $\mathsf{Mult}(\boldsymbol{c_1}, \boldsymbol{c_2})$ is roughly equal to the product of noises of $\boldsymbol{c_1}$ and $\boldsymbol{c_2}$.

The result of Mult is not a valid ciphertext of the basic encryption scheme (a pair) but a triple. Following [BV11,BGV11], one can define an efficient key switching procedure that, given as an input this triple $\boldsymbol{c'}$ (that can be decrypted to some M by using the secret key $(1, s, s^2)^\perp$), outputs a "valid" encryption of M (a pair $\boldsymbol{c''} = (c''_1, c''_2)$) under a different secret/public key pair $(s^*, (\boldsymbol{\psi_1^*}, \boldsymbol{\psi_2^*}))$, such that $\mathsf{Dec}_{s^*}(\boldsymbol{c''}) = M$. In particular, $\boldsymbol{c''}$ has only additively larger noise compared to $\boldsymbol{c'}$.

One can evaluate a $(\wedge, \oplus)$-Boolean circuit homomorphically in a bottom-up manner, by assuming that the inputs are Boolean, and then executing an $\mathsf{Add}(a, \mathsf{Add}(b, \mathsf{Mult}(a, b)))$ at every $a \oplus b$ gate, and a $\mathsf{Mult}(a, b)$ (followed by a key switch) at every $a \wedge b$ gate. (Other gates can implemented by combining AND and XOR.) The main problem is that after a number of Mult steps, the noise has increased so much that the resulting ciphertext will not decrypt correctly. To prevent this, after a number of multiplication steps, one has to do noise reduction.

In the BGV cryptosystem, one can choose between two different reduction mechanisms. The first mechanism, modulus switching, takes a ciphertext modulo p (i.e., from R_p^2) and outputs a ciphertext of the same plaintext modulo some $q \ll p$ (i.e., from R_q^2) such that the noise of the second ciphertext is approximately equal to q/p times the noise of the first plaintext. Thus, applying such modulus switching after every Mult gate, with p/q slightly larger than the noise in fresh ciphertexts, one can keep the noise down almost indefinitely.

Modulus switching has the drawback that the modulus of the inputs to the circuit has to be very large (proportional to the multiplicative complexity of the Boolean circuit), and this means that for circuits of high multiplicative complexity, computational costs will be very large. To circumvent this problem, in addition one can use (after a carefully chosen number of multiplications) bootstrapping, that is, given a ciphertext $C = \mathsf{Enc}_{\mathsf{pk}}(M)$ and homomorphically encrypted bits of a different secret key sk', one can homomorphically decrypt C and obtain the value $\mathsf{Enc}_{\mathsf{pk}'}(M)$ with drastically decreased noise. We refer the reader to [BGV11] for details.

While the resulting (leveled) FHE scheme has relatively large ciphertexts of length $d\lceil \log_2 q\rceil \approx \lambda \log^2 q = \Theta(\lambda \log^2 \lambda)$, it has impressive computational complexity of $\tilde{O}(\lambda^2)$ bit-operations per gate. This can be compared with $\Omega(\lambda^6)$ bit-operations needed in the case of RSA, Paillier's cryptosystem, etc.

Computationally-Private Information Retrieval. In a multi-query m-out-of-n computationally-private information retrieval protocol, (n, m)-CPIR, for ℓ-bit strings, the client has an index set $\boldsymbol{x} = (x_0, \ldots, x_{m-1})$ with $0 \le x_0 < x_1 < \ldots < x_{m-1} \le n-1$ and the server has a database $\boldsymbol{f} = (f_0, \ldots, f_{n-1})$ with $f_i \in \{0,1\}^\ell$. Note that there are $\binom{n}{m}$ such tuples $\boldsymbol{x}$. The client obtains $(f_{x_0}, \ldots, f_{x_{m-1}})$. A two-message (n, m)-CPIR protocol $\mathsf{PIR} = (\mathsf{Query}, \mathsf{Reply}, \mathsf{Answer})$ is *complete* if $\mathsf{Answer}(\ell, \boldsymbol{x}, \mathsf{Reply}(\ell, \boldsymbol{f}, \mathsf{Query}(\ell, \boldsymbol{x}))) = (f_{x_0}, \ldots, f_{x_{m-1}})$ with probability $1 - \kappa^{-\omega(1)}$, or equivalently, with probability $1 - 2^{-\lambda}$. Here, $\mathsf{Query}(\ell, \boldsymbol{x})$ is the client's first message, Reply is the server's second message, and Answer is a local function that the client applies to the reply to obtain back the answer.

Consider the next *chosen index attack* game: First, the attacker selects some index vectors $\boldsymbol{x_0}$ and $\boldsymbol{x_1}$, and sends them to the challenger. Second, the challenger chooses $b \leftarrow \{0,1\}$, and sends $\mathsf{Query}(\ell, \boldsymbol{x}_b)$ back to the attacker. Third, the attacker outputs a bit $b' \in \{0,1\}$. The attacker wins if $b' = b$. We say that an (n, m)-CPIR protocol is *private* if no nonuniform probabilistic polynomial-time adversary wins the chosen index attack game with a non-negligible probability.

The communication complexity of a CPIR protocol is equal to $|\mathsf{Query}|+|\mathsf{Answer}|$. As noted in the introduction, the communication complexity of a trivial non-private information retrieval protocol is equal to $\lceil \log_2 \binom{n}{m}\rceil + m\ell$.

The *rate* of an (n, m)-CPIR protocol is equal to $(\lceil \log_2 \binom{n}{m} \rceil + m\ell)/(|\mathsf{Query}| + |\mathsf{Answer}|)$. The first $(n, 1)$-CPIR protocol with a sublinear communication was proposed in [KO97], and the first protocol with polylogarithmic communication was proposed in [CMS99]. In [Lip05], Lipmaa proposed an $(n, 1)$-CPIR protocol with communication $\Theta(\kappa \cdot \log^2 n + \ell \log n)$, and then in [Lip09] constructed a variation of it with communication $\Theta(\kappa \cdot \log^2 n + \ell)$ and (sublinear and data-dependent) server's computational complexity of $O(n\ell/\log_2(n\ell)/\kappa)$ public-key operations. Both protocols achieve rate 1 when ℓ increases. One can construct an (n, m)-CPIR protocol by batch-coding either of the $(n, 1)$-CPIR protocols, but the result has a subconstant rate.

The first constant-rate $(n, 1)$-CPIR protocol was proposed by Gentry and Ramzan [GR05], and generalized to a constant-rate (n, m)-CPIR protocol by Groth, Kiayias and Lipmaa [GKL10]. More precisely, in the protocol of [GR05], the rate is $1/4$, and in the protocol of [GKL10], the rate is approximately $1/100$. The server's computational complexity of the Groth-Kiayias-Lipmaa protocol is $\Theta(mn\ell/\kappa)$ public-key operations.[4] In all described CPIR protocols, the cost of a single public-key operation is $\Omega(\lambda^6)$.

In [BV11], Brakerski and Vaikuntanathan proposed a new LWE-based fully-homomorphic encryption scheme, and used it to construct an efficient $(n, 1)$-CPIR protocol for 1-bit strings with almost optimal communication and linear computation.

3 Encoding and Decoding of Constant-Weight Codes

Preliminaries. Fix n and m. For $x = x_1 \dots x_n \in \{0,1\}^n$, define its Hamming weight $w_h(x) = \sum x_i = \sharp\{i : x_i = 1\}$. The (n, m) *constant-weight binary code* $\mathsf{C}(n, m)$ is a set of length-n bitstrings such that any $x \in \mathsf{C}(n, m)$ has Hamming weight $w_h(x) = m$. Clearly, $\sharp\mathsf{C}(n, m) = \binom{n}{m}$. Thus, the code C has rate $\lceil \log_2 |\mathsf{C}(n, m)| \rceil / n = \lceil \log_2 \binom{n}{m} \rceil / n$.

There is a natural correspondence between the constant-weight code and the code

$$\mathsf{CC}(n, m) = \{(x_0, \dots, x_{m-1}) : 0 \le x_0 < \dots < x_{m-1} < n \wedge x_i \in \mathbb{Z}\}$$

of length-$(m \cdot \lceil \log_2 n \rceil)$ bitstrings. Unless $m \ge n/\log_2 n$, the codewords of CC are shorter than the codewords of C. Because of this, we say that CC is the *compressed constant-weight code*. Clearly, also $\sharp\mathsf{CC}(n, m) = \binom{n}{m}$.

In the encoding problem for (uncompressed) constant-weight binary codes, one has to construct an efficiently computable bijection between $\mathbb{Z}_{\binom{n}{m}}$ and $\mathsf{C}(n, m)$. The map from $\mathbb{Z}_{\binom{n}{m}}$ to C is called *encoding* (of constant-weight codes), and the map from C to $\mathbb{Z}_{\binom{n}{m}}$ is called *decoding*. The fastest known encoding and decoding algorithms for constant-weight codes (both based on arithmetic coding) were proposed in [Ram90], both requiring a circuit of size $\Theta(n \log n \log\log n \log\log n)$.

Analogously, in the case of compressed constant-weight codes, one has to construct an efficiently computable bijection between $\mathbb{Z}_{\binom{n}{m}}$ and $\mathsf{CC}(n, m)$. The map from $\mathbb{Z}_{\binom{n}{m}}$ to CC is called *encoding* (of compressed constant-weight codes), and the map from CC to $\mathbb{Z}_{\binom{n}{m}}$ is called *decoding*. To achieve efficient encoding and decoding, one sometimes relaxes the previous definition, by replacing $\mathbb{Z}_{\binom{n}{m}}$ with some set $B_m(n)$, elements of which can be represented in $(1 + o(1)) \log_2 \binom{n}{m}$ bits.

Efficient Encoding and Decoding of Compressed Constant-Weight Codes. The fastest known encoding and decoding algorithms for compressed constant-weight codes were recently proposed in [TVS09,SV09], with bit-complexity $O(m^2 \log n)$, which beats the $\Theta(n \log n \log\log n)$ bit-complexity of the algorithm from [Ram90] for $m \le \sqrt{n}$. Moreover, since the algorithms from [Ram90] are for (uncompressed) constant-weight codes, they cannot be directly used in our application. We now give a very short description of the corresponding algorithms, for details see [TVS09,SV09].

First, we need terminology from geometry. Two polytopes P and Q in $\mathbb{R}^m$ are said to be congruent if Q can be obtained from P by a translation, a rotation, and possibly a reflection in a hyperplane. Two polytopes P and Q in $\mathbb{R}^m$ are said to be *equidecomposable* (or equidissectable) if they can be decomposed into finite sets of polytopes $P_1, \dots, P_t$ and $Q_1, \dots, Q_t$, respectively, for some positive integer t, such that P_i and Q_i are congruent for all $i = 1, \dots, t$. That is, P is the disjoint union of the (possibly reflected) polytopes P_i, and similarly for Q. Then we say that P can be dissected to give Q, and vice versa.

[4] We note that while the Groth-Kiayias-Lipmaa protocol implicitly uses an efficient (according to our terminology) computational batch code, it is used to bring down the server's computational complexity to linear in $\Theta(mn)$.

The code $\mathsf{CC}(n,m)$ can be seen as a unit ortoscheme in m-dimensional vector space. For example, $\mathsf{CC}(n,2)$ is an upper triangle, bounded by points $(0,1)$, $(n-1,n-1)$ and $(0,n-1)$ in $\mathbb{R}^2$ (see Fig. 1). On the other hand, let $B_m(n)$ be the polytope $B_m(n) := [0,n)\times[n-n/2,n)\times\cdots\times[n-n/m,n)$. (The upper rectangle, or "brick" in Fig. 1.). Note that the volume of both $\mathsf{CC}(n,m)$ and $B_m(n)$ is $\frac{n^m}{m!}$. A point in $B_m(n)$ is presented as $(y_0,\ldots,y_{m-1})$ with $y_i \in [(n-1)-(i+1)/m, n-1)$, and thus can be encoded by $\leq \log_2 \binom{n}{m} + m$ bits. Clearly, $\mathsf{CD}(n,2)$ and $B_2(n)$ are equidecomposable. This can be achieved by translating the polytope 1 up and right, and then reflecting it.

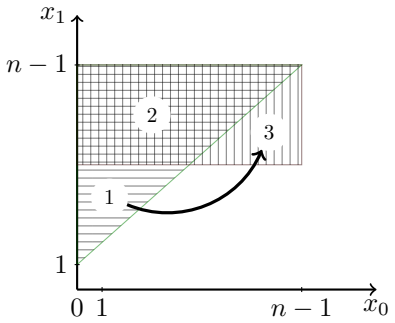

Fig. 1. $\mathsf{CD}(n,2)$ and $B_2(n)$ are equidecomposable

For $m \leq 4$, two polytopes are equidecomposable if and only if they have the same volume and the same Dehn invariant. For $m > 4$, it is only known that equality of Dehn invariants is a necessary condition. However, for $\mathsf{CC}(n,m)$ and $B_m(n)$, [TVS09] was able to efficiently dissect $\mathsf{CC}(n,m)$ to give $B_m(n)$ (the decoding problem) and vice versa (the encoding problem).

Briefly, both dissections of [TVS09] are recursive, first constructing dissections between $(m-1)$-dimensional subspaces, and then efficiently handling the last coordinate of the original polytope. Inside a recursion step, one performs $\Theta(m)$ addition/subtraction/comparison operations on $\Theta(\log n)$-bit operands, and thus the total computational complexity of the Tian-Vaishampayan-Sloane algorithm [TVS09] is $\Theta(m^2 \log n)$ bit operations. Moreover, the Tian-Vaishampayan-Sloane algorithm can be described by a circuit of size $\Theta(m^2 \log n)$. Finally, the dissection from [SV09] is easier than the one from [TVS09], which also means that the resulting algorithm is simpler; however the bit-complexity of the algorithm from [SV09] remains the same, $\Theta(m^2 \log n)$.

4 Efficient Parallel Computation

Circuits and Circuit Complexity. Let $L(f)$ be the circuit size complexity of some function $f : \{0,1\}^\nu \to \{0,1\}^\ell$. Let $L(\nu,\ell)$ be the circuit size complexity of the hardest Boolean function $f : \{0,1\}^\nu \to \{0,1\}^\ell$. We denote $L(\nu) := L(\nu,1)$. The most precise known bound for $L(\nu)$ (originating from early work of Shannon and Lupanov [Lup58]) appears to be $L(\nu) \leq \frac{2^\nu}{\nu}\cdot\left(1+3\cdot\frac{\log\nu}{\nu}+O\left(\frac{1}{\nu}\right)\right)$. For an arbitrary $f : \{0,1\}^\nu \to \{0,1\}^\ell$, Lupanov [Lup58] has established a less-known upperbound $L(\nu,\ell) = (1+o(1))2^\nu\ell/\log_2(2^\nu\ell)$.

Parallel Computation: Uhlig's Upper Bound. Let $L_m(\nu,\ell)$ be the upperbound on the circuit size complexity of parallel evaluation of any function $f : \{0,1\}^\nu \to \{0,1\}^\ell$ on any m (independent) inputs. As shown by Uhlig [Uhl92], given the mentioned upperbound on $L(\nu,1)$, one can derive a similar upperbound on $L_m(\nu,1)$ for any $m < 2^{\nu/\log_2\nu}$. In Uhlig's construction, one fixes a suitable k, and then by letting $0 \leq i < 2^k$ be the binary presentation of $(x_{\nu-k+1},\ldots,x_\nu) \in \{0,1\}^k$ and setting $f_i(x_1,\ldots,x_{\nu-k}) := f(x_1,\ldots,x_\nu)$, defines

$$\begin{cases} g_0(x_1,\ldots,x_{\nu-k}) := f_0(x_1,\ldots,x_{\nu-k}) \ , \\ g_i(x_1,\ldots,x_{\nu-k}) := f_{i-1}(x_1,\ldots,x_{\nu-k}) \oplus f_i(x_1,\ldots,x_{\nu-k}) \ , \quad 0 < i < 2^k \ , \\ g_{2^k}(x_1,\ldots,x_{\nu-k}) := f_{2^k}(x_1,\ldots,x_{\nu-k}) \ . \end{cases}$$

Now, let $0 \leq j < 2^k$ be the binary presentation of $(y_{\nu-k+1},\ldots,y_\nu)$. If say $i < j$, then $f(x_1,\ldots,x_\nu) = f_i(x_1,\ldots,x_{\nu-k}) = g_0(x_1,\ldots,x_{\nu-k}) \oplus \cdots \oplus g_i(x_1,\ldots,x_{\nu-k})$, while $f(y_1,\ldots,y_\nu) = f_j(y_1,\ldots,y_{\nu-k}) = g_{j+1}(y_1,\ldots,y_{\nu-k}) \oplus \cdots \oplus g_{2^k}(y_1,\ldots,y_{\nu-k})$.

Thus, to compute both $f(x_1,\ldots,x_\nu)$ and $f(y_1,\ldots,y_\nu)$, one evaluates 2^k+1 "small" functions $g_t : \{0,1\}^{\nu-k} \to \{0,1\}$ for $0 \leq t \leq 2^k$, each of them having circuit size complexity $(1+o(1))2^{\nu-k}/(\nu-k)$ due to Lupanov's bound on $L(\cdot)$. For $k = \lceil \log_2\nu - \frac{1}{2}\cdot\log_2\log_2\nu \rceil$, all 2^k+1 functions g_t can be evaluated within total circuit size complexity $(1+o(1))2^\nu/\nu$. On top of it, one also has to evaluate a number of multiplexers (to decide whether g_t gets $(x_1,\ldots,x_{\nu-k})$ or $(y_1,\ldots,y_{\nu-k})$ as an input), a number of conditional XORs (to decide which outputs of which g_t-s will be XORed together) and a small number of other gates, all together in $o(2^\nu/\nu)$ gates. Evaluating f on 2^t inputs, for any $t > 1$, can be done recursively by using the same idea as long as $t = o(\nu/\log\nu)$ or $m = o(2^{\nu/\log\nu})$.

Generalization of Uhlig's Upper Bound to Non-Boolean f. Let $f : \{0,1\}^\nu \to \{0,1\}^\ell$. We use a similar construction as in the Boolean case, but assuming that all functions f, f_i and g_i have range $\{0,1\}^\ell$. Then by Lupanov's upper bound, each $g_t : \{0,1\}^{\nu-k} \to \{0,1\}^\ell$ can be evaluated by a circuit of size $(1+o(1))2^{\nu-k}\ell/\log_2(2^{\nu-k}\ell)$.

Setting $k = \lceil \log_2 \nu - \frac{1}{2} \cdot \log_2 \log_2 \nu \rceil$ as before, we get that all $2^k + 1$ functions g_t can be evaluated in total circuit size $(1 + o(1))2^\nu \ell / \log_2(2^\nu \ell)$. Since this clearly dominates the circuit size, we have proven the next theorem.

Theorem 1. *Let $f : \{0,1\}^\nu \to \{0,1\}^\ell$. Let $m < 2^{\nu / \log_2 \nu}$. Then f can be evaluated in parallel on m different inputs, in a total circuit size $(1 + o(1))2^\nu \ell / \log_2(2^\nu \ell)$.*

5 Computational Batch Codes

In [IKOS04], the authors defined (information-theoretical) batch codes. However, their constructions are not sufficiently efficient for our purposes. We weaken their notion by defining computational batch codes, and then show that one can construct computational batch codes with essential optimal parameters.

Informally, in a (computational) $[n, N, m, M, T]_\ell$-batch code, the server stores a database $\boldsymbol{f} = (f_0, \ldots, f_{n-1})$ of ℓ-bit strings. He divides the database into M bins, all together containing N (ℓ-bit) strings, such that when he later needs to retrieve any m (ℓ-bit) strings from this database, it can do it by querying up to T (ℓ-bit) strings from any bin and then recombining the results. For the sake of simplicity, assume that $M \mid N$, and that all bins contain N/M elements.

Definition 1 (Computational Batch Codes). *An $[n, N, m, M, T]_\ell$-*computational batch code for ℓ-bit strings *is a tuple of efficient algorithms* $(\mathsf{BCGen}, \mathsf{BCEncode}, \mathsf{BCRetrieve})$ *with the following properties:*

- *The key generation algorithm* $\mathsf{BCGen}(1^\lambda)$ *outputs a key s.*
- *For $\boldsymbol{f} = (f_0, \ldots, f_{n-1})$ with $f_i \in \{0,1\}^\ell$, $\mathsf{BCEncode}_s(\boldsymbol{f}) = (\boldsymbol{f'_0}, \ldots, \boldsymbol{f'_{M-1}})$, such that $\boldsymbol{f'_i} = (f'_{i0}, \ldots, f'_{i,(N/M)-1})$ and $f'_{ij} \in \{0,1\}^\ell$. Here, $\boldsymbol{f'_i}$ is the value in the ith bin.*
- *For $\boldsymbol{x} = (x_0, \ldots, x_{n-1})$ with $0 \le x_0 \le \cdots \le x_{n-1}$, $\mathsf{BCRetrieve}_s((\boldsymbol{f'_0}, \ldots, \boldsymbol{f'_{M-1}}), \boldsymbol{x})$ makes no more than T probes to each bin, and then returns a tuple $\boldsymbol{a} = (a_0, \ldots, a_{n-1})$ with $a_j \in \{0,1\}^\ell$. We assume that* $\mathsf{BCRetrieve}$ *outputs $\perp$ (fail) if it tries to access more than T probes from some $\boldsymbol{f'_i}$.*

Definition 2 (Completeness of Computational Batch Codes). *A computational batch code is* $(1 - \varepsilon)$-complete *if for any $\boldsymbol{f}$ and $\boldsymbol{x}$,* $\Pr[s \leftarrow \mathsf{BCGen}(1^\kappa), \mathsf{BCRetrieve}_s(\mathsf{BCEncode}_s(\boldsymbol{f}), \boldsymbol{x}) = (f_{x_0}, \ldots, f_{x_{m-1}})] \ge 1 - \varepsilon$.

We say that n/N is *the rate* of the batch code. In [IKOS04], the authors proposed several 1-complete batch codes. In particular, they showed that batch codes cannot exist if $M < m/T$. They also showed that rate 1 is generally unachievable; their best constructions achieved rate $1 - \varepsilon$ with $M = \mathrm{poly}(m)$ (with $M = \Omega(m^2)$; this is since in the subset codes of [IKOS04], $M = \Omega(m^{H(\alpha)/\alpha})$, where $0 < \alpha < 1/2$), and rate $1/\log n$ with $M = O(m)$.

Next, we describe a computational batch code that is loosely based by an implicit construction of Groth, Kiayias and Lipmaa from [GKL10] who however did not use the terminology of batch codes. We show that for certain parameters, this *CB* computational batch code has rate 1 with almost optimal $M = (1 + \varepsilon)K/T$ for any $\varepsilon > 0$, at the same time providing overwhelming completeness $1 - \kappa^{-\omega(1)}$ (or, exponential completeness $1 - 2^{-\lambda}$), given a mild lowerbound on the value of T.

Recall that pseudorandom generator $P : \{0,1\}^\lambda \to \{0,1\}^L$, where $L(\lambda) > \lambda$, is a non-uniform probabilistic polynomial-time algorithm, such that for an arbitrary non-uniform probabilistic polynomial-time adversary A, $\Pr[x \leftarrow \{0,1\}^\lambda : A(1^\lambda, P(1^\lambda, x)) = 1] - \Pr[y \leftarrow \{0,1\}^L : A(1^\lambda, y) = 1] = \kappa^{-\omega(1)}$. Alternatively, one can assume that we have an exponentially secure pseudorandom generator with the last probability being upperbounded by $2^{-\lambda}$. If $L(\lambda) \le \lambda$, then P will just output its input.

We are now ready to describe the CB computational $[n, n, m, M = \lceil m/T \rceil, (1 + \varepsilon)T]_\ell$-batch code for ℓ-bit strings, where $P : \{0,1\}^\lambda \to \{0,1\}^{\lceil \log_2 \binom{n}{m} \rceil + m}$ is a fixed pseudorandom generator. Basically, in this batch code, the order of database elements is permuted according to the output of P on a fixed seed s, and in the retrieval phase, the server access the elements according to the nonpermuted order.

- $\mathsf{BCGen}(1^\lambda)$ outputs a new random seed $s \leftarrow \{0,1\}^\lambda$. From $P(s)$, the participants compute a permutation $\pi : [n] \to [n]$ by using the compressed constant-weight code decoding algorithm from [SV09].
- For $i \in \{0, \ldots, M-1\}$, $\mathsf{BCEncode}_s(\boldsymbol{f})$ puts to the ith bin the tuple $\boldsymbol{f'_i} := (f'_{i,0}, \ldots, f'_{i,(n/M)-1})$, where $f'_{i,j} := f_{\pi(i(n/M)+j)}$. Let $\boldsymbol{f'} = (f'_0, \ldots, f'_{M-1})$.
- $\mathsf{BCRetrieve}_s(\boldsymbol{f'}, \boldsymbol{x})$ does the following.
  1. For every $x_i \in \{0, \ldots, n-1\}$, $\mathsf{BCRetrieve}_s$ makes a query $\pi^{-1}(x_i \bmod N/M)$ to the bin $\pi^{-1}(\lfloor x_i/(N/M) \rfloor)$.

2. $\mathsf{BCRetrieve}_s$ returns $\perp$ if it has made more than $(1+\varepsilon)T$ queries to some bin i, otherwise it returns all query results in a correct order.

Theorem 2. *Assume that the indexes $\pi(i)$ are randomly distributed. Fix $\varepsilon > 0$. The CB computational $[n, n, m, M = \lceil 2m/T \rceil, (1+\varepsilon)T]_\ell$-batch code is complete with probability $1 - \frac{m}{T} \cdot e^{-2\alpha_{n,m}(\varepsilon T)^2}$ where $\alpha = \alpha_{n,m} = n(n-m+1)/m$.*

Proof. Let the bin size be $\beta = \lceil n/M \rceil \leq \lceil Tn/m \rceil$. To bound the failure probability, we first bound the probability that among randomly chosen indexes $\pi(x_0), \ldots, \pi(x_{n-1})$, there are more than $(1+\varepsilon)T$ that fall into the same (fixed) β-element bin. Let $X = X(m, n, \beta)$ be a hypergeometric random variable describing the process of counting how many white balls are selected when m balls are randomly selected without replacement from a population of n balls of which β are white. Clearly, the mean of X is $m\beta/n$. Thus, we first have to compute the probability $\Pr[X \geq (1+\varepsilon)T]$. But by [Ser74], $\Pr[X - E[X] > \gamma] < e^{-2\alpha\gamma^2}$. In our application, $E[X] = T$, so $\Pr[X > (1+\varepsilon)T] = \Pr[X - T > \varepsilon T] = \Pr[X - E[X] > \varepsilon T] < e^{-2\alpha(\varepsilon T)^2}$. Thus, the probability that no bin has more than $(1+\varepsilon)T$ balls is $(1 - e^{-2\alpha(\varepsilon T)^2})^M \geq 1 - M \cdot e^{-2\alpha(\varepsilon T)^2} \geq 1 - \frac{m}{T} \cdot e^{-2\alpha(\varepsilon T)^2}$. □

In particular, we achieve completeness $1 - 2^{-\lambda}$ if

$$T \geq \sqrt{m\lambda \cdot \ln 2}/(\varepsilon\sqrt{2n(n-m+1)}) \ . \tag{1}$$

To bound the error probability for fixed n, we note that the minimum value of $\alpha_{m,n}$ for this n is $\alpha_{m,n} = 2$. Thus, we get that the CB computational batch code is $(1 - \varepsilon_p - e^{-4(\varepsilon T)^2})$-complete, independently of the value of m. Thus in this case[5], we can take $T \geq \frac{\lambda \ln 2}{\varepsilon\sqrt{2}}$. In the rest of the paper, we will use the value $\varepsilon = 1$. This considerably sharpens the less precise Chernoff-based bound $T > 3 \ln 2 \cdot \lambda$ of [GKL10] and thus may again be of independent interest.

Finally, assume that P is a pseudorandom generator that is (say) $(1 - 2^{-\lambda-10})$-secure against efficient adversaries. Then the CB computational batch code is still $(1 - 2^{-\lambda})$-complete under the constraint Eq. (1).

6 Improving the Rate

A straightforward implementation of any FHE-based protocol has a rate $1/\kappa$, i.e., per 1 bit of useful data, κ bits are exchanged. This is since every input to the circuit and output of the circuit is encrypted bitwise. With the BGV cryptosystem, $\kappa = \Theta(\lambda \log^2 \lambda)$ is relatively large.

In this section, we briefly discuss how to achieve a better rate without significantly increasing the computation. First, we show how to compress client's message from rate $1/\kappa$ to rate 1, and second, we show how to compress server's message from rate $1/\kappa$ to rate $1 - o(1)$. The latter compression assumes the use of the BGV cryptosystem, though it also works with some other alternatives.

6.1 Compressing Client's Message

We use the next idea, up to our knowledge first presented by Adam Smith at the Asiacrypt 2009 rump session [HPS10] and then used for the purposes of CPIR in [BV11]: the client homomorphically encrypts the secret key symk (of bitlength λ) of a CPA-secure symmetric cipher E (e.g., a stream cipher or AES in the counter mode, or a custom-designed symmetric cryptosystem that is based on the same assumption as the FHE, see [BV11]), and sends this together with E-encrypted inputs to the server. Assuming that the client's private input is L bit long (in our case, $L \approx \log \binom{n}{m}$), this means that the client has to send $L + \Theta(\lambda\kappa)$ bits. Given a homomorphically encrypted secret key symk and a symmetrically encrypted input L, the server computes homomorphically encrypted inputs by homomorphically evaluating a circuit for E. In our case, the client inputs are $\log_2 \binom{n}{m}$ bits long, and thus the work of both the client and the server is increased by $\Theta(\log \binom{n}{m} \cdot \lambda)$, assuming that it takes $\Theta(\lambda)$ time to evaluate E once. After that, the server runs the $1/\kappa$-rate CPIR protocol, and thus computes $m\ell$ ciphertexts.

[5] This also means that the computational batch code cannot be used if $m < 0.6\sqrt{\lambda}$. However, then in Sect. 7 we can just use the restricted CPIR protocol without batch-coding, unless simultaneously $m > \sqrt{n/\log n}$. But in the case $\sqrt{n/\log n} < 0.6\sqrt{\lambda}$ the database size is so small that one can use parallel repetition of the restricted CPIR. Second, here we need that $m < n/2$. If $m \geq n/2$ then the whole database can just be transfered to the client.

$\mathsf{Query}(\ell, \boldsymbol{x})$**:** The client executes the next steps.

1. He generates a new symmetric key symk, a new public/secret key $(\mathsf{pk}, \mathsf{sk})$ pair for FHE scheme, and a seed s for the batch code.
2. According to his input $\boldsymbol{x}$, he forms in parallel $\lceil m/T \rceil$ restricted $(\beta, 2T)$-CPIR protocols to different $\lceil m/T \rceil$ bins as defined by $\mathsf{BCEncode}_s$. If some bin contains too many elements of $\boldsymbol{f}$, then the client aborts. Otherwise, let $\boldsymbol{x'_i} = (x'_{i0}, \ldots, x'_{i,2T-1})$ be his query to to the ith bin. For every i, the client decodes the constant-weight codeword $(x'_{i,0}, \ldots, x'_{i,2T-1})$ to a $\lceil \log_2 \binom{\beta}{T} \rceil$-bit string y_i.

He sends to the server the next values: 1) the new public key pk, 2) homomorphic bitwise encryption of symk by using the public key pk, 3) the seed s, and 4) symmetric encryption y_e of $(y_0, \ldots, y_d)$ by using the key symk.

$\mathsf{Reply}(\ell, \boldsymbol{f}, \mathsf{Query})$**:** The server executes the next steps.

1. She homomorphically decrypts y_e, by using pk and a homomorphic bitwise encryption of symk.
2. She homomorphically encodes, by using the key pk, the result as a homomorphic encryption of constant-weight codewords $(x_{i,0}, \ldots, x_{i,2T-1})$.
3. She uses $\mathsf{BCRetrieve}_s$ to compute circuits for $\lceil m/T \rceil$ functions f'_i identified with their truth tables $\boldsymbol{f'_i} = (f'_{i0}, \ldots, f'_{i,2T-1})$. More precisely, to construct the circuit for function $\boldsymbol{f'_i}$, the server divides the corresponding $2T$-element subdatabase into further $n^{1/\log_2 \log_2 n}$-element subdatabases. For every latter subdatabase, the server constructs a parallel circuit, as described in Sect. 4. The circuit for $\boldsymbol{f'_i}$ consists of $\beta/n^{1/\log_2 \log_2 n}$ parallel such circuits.
4. She packs resulting $2m\ell$ ciphertexts into $\lceil 2m\ell/d \rceil$ ciphertexts, where d is the lattice dimension

She sends the resulting $\lceil 2m\ell/d \rceil$ ciphertexts to the client.

$\mathsf{Answer}(\ell, \boldsymbol{x}, \mathsf{Reply})$**:** The client decrypts server's reply, that is, all $\lceil 2m\ell/d \rceil$ ciphertexts. He unpacks them to $2m\ell$ plaintext bits, and then outputs $2m$ ℓ-bit strings.

Fig. 2. The new (n, m)-CPIR protocol

6.2 Compressing Server's Message

Recall that the plaintext space of the BGV cryptosystem is R_2, while the ciphertext space is R_q^2. In particular, all our computation is made in $\mathbb{Z}_2 \subset R_2$. As mentioned in [BGV11], one can use a larger plaintext space R_p for a prime p, where p is polynomial in the security parameter, $p = \mathrm{poly}(\lambda)$. We are not going to apply homomorphic operations on R_p (this may run in complications due to the fact that p is splits completely to prime ideals in R, see [BV11]). We emphasize that the next idea by itself is not novel.

Instead, we note that in a FHE-based (n, m)-CPIR protocol the server returns to the client many different ciphertexts. We combine $d\lfloor \log p \rfloor$ ciphertexts C_{ij}, $i \in [p]$ and $j \in [d]$ to one as $C \leftarrow \sum_{j=1}^{d} (\sum_{i=1}^{\lfloor \log p \rfloor} 2^i C_{ij}) x^j$. While every addition increases the noise additively, we can assume that the final noise is not too big, and thus one can successfully decrypt C. Since the BGV cryptosystem is homomorphic, C decrypts to M, where $M = \sum_j (\sum_i 2^i M_{ij}) x^j$ and M_{ij} is the plaintext corresponding to C_{ij}.

Now recall that the size of a ciphertext is $d \log q$, and the size of the "plaintext" is $d \log p$. According to [BGV11], one can assume that $\log q = \Theta(\lambda)$, $d = \Omega(\lambda \log \lambda)$. Our own calculations show that one can choose $q = \tilde{O}(p) \cdot \tilde{O}(\lambda^{3/2})$ where in both cases, the degree of $\log$ in $\tilde{O}$ is small. Thus, $\log p = \log q - (3/2 + o(1)) \log \lambda$. Therefore, the rate of this "batched" BGV cryptosystem is $\approx d \log p / (d \log q)$ that can be arbitrarily close to 1.

7 New m-out-of-n CPIR Protocol

We already gave a very high-level step-by-step derivation of the new rate $1 - o(1)$ CPIR protocol in the introduction. In the current section, we give a concise description of the final protocol.

For simplicity, we assume that $\ell \leq \kappa$, otherwise we can just repeat the next protocol $\lceil \ell/\kappa \rceil$ times. We use a symmetric cryptosystem (e.g., AES) and an FHE scheme (e.g., the BGV cryptosystem). Assume as always that the client's input is an m-tuple $(x_0, \ldots, x_{m-1})$, and that the server's input is a database $\boldsymbol{f} = (f_0, \ldots, f_{n-1})$ also expressed as a function $f : \mathbb{Z}_n \to \{0,1\}^\ell$ with $f(i) = f_i$. The client needs to retrieve $(f_{x_0}, \ldots, f_{x_{m-1}})$. Let $(\mathsf{BCGen}, \mathsf{BCEncode}, \mathsf{BCRetrieve})$ be a computational $[n, n, m, M = \lceil m/T \rceil, 2T]_\ell$ batch code for some fixed T. Let $\beta \leftarrow \lfloor Tn/m \rfloor$. The full protocol is depicted by Fig. 2.

Theorem 3. *Assume that P is a secure pseudorandom generator and that one uses the BGV cryptosystem. If T satisfies Eq.* (1)*, then the new CPIR protocol is $(1 - 2^{-\lambda})$-complete.*

Proof. First, due to Thm. 2, we get that if T is big enough, then the first chosen s is good with overwhelming probability. Since in this case we choose a new s, we get a perfectly complete protocol by doing an expected number 1 of choices. Second, due to the second assumption, as specified in Sect. 6, the client can correctly recover from every server's ciphertext c a d-bit plaintext. □

Note that the larger is T, the higher is the completeness probability.

Theorem 4. *Assume that the used symmetric cryptosystem is CPA-secure, and that the used FHE scheme is CPA-secure and KDM-secure. Then the new (n, m)-CPIR protocol is private.*

Proof. The server only sees data encrypted by either the symmetric or the FHE scheme. □

Theorem 5. *Excepting the costs that do not depend on n, m or ℓ, the communication complexity of the new (n, m)-CPIR protocol is $\lceil \log_2 \binom{n}{m} \rceil + (1 + o(1))m\ell + \Theta(\lambda\kappa)$.*

Proof. Assuming $T \mid m$, the client's relevant message is a symmetric encryption of his $\log_2 \binom{Tn/m}{T} \cdot m/T \approx \log_2 \binom{n}{m}$ input bits together with FHE-encryption of the secret key bits of the symmetric cryptosystem. The server's message is a homomorphic encryption of his $2m\ell$ output bits. Due to the technique of Sect. 6, her message has length $(1 + o(1))m\ell$. □

Theorem 6. *Assume that the symmetric primitive on plaintext of t bits can be decrypted by a circuit of size $\mathsf{T}_{\mathsf{sym}}(t)$. Assume that we are using the BGV cryptosystem. The server's computational complexity is dominated by*

$$\left(\frac{(2 + o(1))Tn\ell}{(Tn/m)^{1/\log_2 \log_2 (Tn/m)} \log_2(Tn\ell/m)} + \mathsf{T}_{\mathsf{sym}} \left(\log_2 \binom{n}{m} \right) \right) \cdot \tilde{O}(\lambda^2)$$

bit-operations, where $T \geq \sqrt{m\lambda \cdot \ln 2}/(\sqrt{2n(n - m + 1)})$.

Proof. The first addend in the first multiplicand follow from the estimation given in introduction. More precisely, a restricted (n, m)-CPIR requires $rcomm(n, m, \ell) = (1 + o(1))nm\ell/(n^{1/\log_2 \log_2 n} \cdot \log_2(n\ell))$ public-key operations. Therefore, the full rate-$1/\kappa$ (n, m)-CPIR protocol requires

$$\begin{aligned} \frac{m}{T} \cdot rcomp(Tn/m, 2T, \ell) &= \frac{m}{T} \cdot \left(\frac{(1 + o(1))Tn/m \cdot 2T \cdot \ell}{(Tn/m)^{1/\log_2 \log_2 (Tn/m)} \log_2((Tn/m)\ell)} \right) \\ &= \frac{(2 + o(1))Tn\ell}{(Tn/m)^{1/\log_2 \log_2 (Tn/m)} \log_2(Tn\ell/m)} \end{aligned}$$

public-key operations which is optimized if $T = \sqrt{m\lambda \cdot \ln 2}/(\sqrt{2n(n - m + 1)})$. The second addend is straightforward. The second multiplicand $\tilde{O}(\lambda^2)$ comes from the estimation of [BGV11]. □

Clearly, the server's computational complexity is minimized when T obtains the smallest allowed value. E.g., $T = 6$ in practice, although $T = 1$ suffices for $m \leq 2n(1 + n)/(2n + \lambda \ln 2)$. As an example, if $n = 10^6$ and $\lambda = 80$ then $T = 1$ suffices for $m \leq 10^6 - 27$.) Finally, note that the computation is dominated by $\mathsf{T}_{\mathsf{sym}}(\log_2 \binom{n}{m})$, and thus in practice it is probably advantageous not to use the optimization of [HPS10].

Acknowledgments. The author was supported by Estonian Science Foundation, grant #8058, and European Union through the European Regional Development Fund.

References

[BGV11] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully Homomorphic Encryption without Bootstrapping. Technical Report 2011/277, International Association for Cryptologic Research, May 28, 2011. Available at http://eprint.iacr.org/2011/277. Version from August 11, 2011.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient Fully Homomorphic Encryption from (Standard) LWE. In *FOCS 2011*, pages ?–?, Palm Springs, CA, USA, October, 23–25 2011. IEEE, IEEE Computer Society Press. To appear.

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private Information Retrieval. In *FOCS 1995*, pages 41–50, Milwaukee, Wisconsin, USA, October 23–25 1995. IEEE.

[CMS99] Christian Cachin, Silvio Micali, and Markus Stadler. Computational Private Information Retrieval with Polylogarithmic Communication. In Jacques Stern, editor, *EUROCRYPT 1999*, volume 1592 of *LNCS*, pages 402–414, Prague, Czech Republic, May 2–6, 1999. Springer-Verlag.

[Gen09a] Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford, September 2009.

[Gen09b] Craig Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In Michael Mitzenmacher, editor, *STOC 2009*, pages 169–178, Bethesda, Maryland, USA, May 31 — June 2, 2009. ACM Press.

[GKL10] Jens Groth, Aggelos Kiayias, and Helger Lipmaa. Multi-Query Computationally-Private Information Retrieval with Constant Communication Rate. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 107–123, Paris, France, May 26–28, 2010. Springer-Verlag.

[GR05] Craig Gentry and Zulfikar Ramzan. Single-Database Private Information Retrieval with Constant Communication Rate. In Luis Caires, Guiseppe F. Italiano, Luis Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP 2005*, volume 3580 of *LNCS*, pages 803–815, Lisboa, Portugal, 2005. Springer-Verlag.

[HPS10] Sean Hallgren, Chris Peikert, and Adam Smith. Personal communication. Also presented in Asiacrypt 2009 rump session, January 2010.

[IKOS04] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Batch Codes And Their Applications. In *STOC 2004*, pages 262–271, Chicago, IL, USA, June 13–16 2004. ACM Press.

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. Replication is Not Needed: Single Database, Computationally-Private Information Retrieval. In *FOCS 1997*, pages 364–373, Miami Beach, Florida, October 20–22, 1997. IEEE Computer Society.

[Lip05] Helger Lipmaa. An Oblivious Transfer Protocol with Log-Squared Communication. In Jianying Zhou and Javier Lopez, editors, *ISC 2005*, volume 3650 of *LNCS*, pages 314–328, Singapore, September 20–23, 2005. Springer-Verlag.

[Lip08] Helger Lipmaa. New Communication-Efficient Oblivious Transfer Protocols Based on Pairings. In Tzong-Chen Wu, Chin-Laung Lei, Vincent Rijmen, and Der-Tsai Lee, editors, *ISC 2008*, volume 5222 of *LNCS*, pages 441–454, Taipei, Taiwan, September 15–18 2008. Springer-Verlag.

[Lip09] Helger Lipmaa. First CPIR Protocol with Data-Dependent Computation. In Donghoon Lee and Seokhie Hong, editors, *ICISC 2009*, volume 5984 of *LNCS*, pages 193–210, Seoul, Korea, December 2–4, 2009. Springer-Verlag.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On Ideal Lattices and Learning with Errors over Rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23, Nice, France, May 30–June 3, 2010. Springer-Verlag.

[Lup58] Oleg B. Lupanov. The Synthesis of Contact Circuits. *Dol. Akad. Nauk SSR (N.S.)*, 119:23–26, 1958.

[MPSW05] Silvio Micali, Chris Peikert, Madhu Sudan, and David A. Wilson. Optimal Error Correction Against Computationally Bounded Noise. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 1–16, Cambridge, MA, USA, February 10–12, 2005. Springer-Verlag.

[Ram90] Tenkasi V. Ramabadran. A Coding Scheme for m-out-of-n Codes. *IEEE Transactions on Communications*, 38(8):1156–1163, August 1990.

[Ser74] Robert J. Serfling. Probability Inequalities for the Sum in Sampling without Replacement. *The Annals of Statistics*, 2(1):39–48, 1974.

[SV09] Neil J. A. Sloane and Vinay A. Vaishampayan. Generalizations of Schöbi's Tetrahedral Dissection. *Discrete & Computational Geometry*, 41(2):232–248, 2009.

[TVS09] Chao Tian, Vinay A. Vaishampayan, and Neil J. A. Sloane. A Coding Algorithm for Constant Weight Vectors: A Geometric Approach Based on Dissections. *IEEE Transactions on Information Theory*, 55(3):1051–1060, 2009.

[Uhl74] Dietmar Uhlig. On the Synthesis of Self-Correcting Schemes from Functional Elements with a Small number of Reliable Elements. *J. Math. Notes Acad. Sci. USSR*, 15:558–562, 1974.

[Uhl92] Dietmar Uhlig. Networks Computing Boolean Functions for Multiple Input Values. In M. S. Paterson, editor, *Boolean Function Complexity*, volume 169 of *London Mathematical Society*, pages 163–173. Cambridge University Press, 1992.