XSPIR: Efficient Symmetrically Private Information Retrieval from Ring-LWE

Chengyu Lin(✉), Zeyu Liu, and Tal Malkin

Columbia University, New York, NY 10027, USA
{chengyu,tal}@cs.columbia.edu, zl2967@columbia.edu

Abstract. Private Information Retrieval (PIR) allows a client to retrieve one entry from a database held by a server, while hiding from the server which entry has been retrieved. Symmetrically Private Information Retrieval (SPIR) additionally protects the privacy of the data, requiring that the client obtains only its desired entry, and no information on other data entries.

In recent years, considerable effort has been expanded towards making PIR practical, reducing communication and computation. State-of-the-art PIR protocols are based on homomorphic encryption from the ring-LWE assumption. However, these efficient PIR protocols do not achieve database privacy, and leak a lot of information about other data entries, even when the client is honest. Generic transformation of these PIR protocols to SPIR have been suggested, but not implemented.

In this paper, we propose XSPIR, a practically efficient SPIR scheme. Our scheme is based on homomorphic encryption from ring-LWE like recent PIR works, but achieves a stronger security guarantee with low performance overhead. We implement XSPIR, and run experiments comparing its performance against SealPIR (Angel et al., IEEE S&P 2018) and MulPIR (Ali et al., USENIX SECURITY 2021). We find that, even though our scheme achieves a stronger security guarantee, our performance is comparable to these state-of-the-art PIR protocols.

Keywords: Private information retrieval · Symmetrically private information retrieval · Homomorphic encryption · Ring-LWE

1 Introduction

Private information retreival (PIR) [19] allows a client to retrieve a data entry from a server, while hiding which entry was retrieved from the server. In this paper we focus on the single-server setting, which requires computational security (sometimes referred to as computational PIR, or cPIR). This is an important building block which can benefit many privacy preserving applications, including private media steaming [2,38], subscription [18], private group messaging [17], anonymous communication [6,37,44,53], and ad delivery [36]. However, PIR is quite costly. First, it requires the server to process the whole database in order to maintain the privacy of the query. This is inherent, since if certain entries

V. Atluri et al. (Eds.): ESORICS 2022, LNCS 13554, pp. 217–236, 2022.
https://doi.org/10.1007/978-3-031-17140-6_11

are not processed, the server would know they are not retrieved by the client. Moreover, existing schemes require a much higher overhead in computation and communication, depending on a cryptographic security parameter.

There are many PIR protocols in the literature [1,2,12,25,26,30,33,41,43,46,61], including considerable implementation efforts in recent years. These implementations follow two lines. The first [21,22,55] follows an approach introduced by Gentry and Ramzan [33], which has good communication but a high computational cost. The second approach builds on XPIR by Aguilar-Melchor et al. [2], based on homomorphic encryption. This approach includes the most efficient implementations to date: SealPIR by Angel et al. [5], MulPIR by Ali et al. [4], and SHECS-PIR by Park et al. [57].

A stronger version of PIR is *Symmetrically Private Information Retrieval (SPIR)* [34], where we additionally require privacy for the server's data. Specifically, the requirement is that the client should only learn the retrieved data entry, but not any information about any other data entries. This can be useful in many applications where the data consists of sensitive information (e.g., a medical database).

Currently, all these implemented PIR schemes do not satisfy such a security guarantee. For the homomorphic encryption based protocols (the line we will follow in this paper), the reason is the following. To improve efficiency, these protocols take advantage of compressing more data into a single ciphertext, allowing the client to retrieve a large chunk of data from each ciphertext (usually more than one entry). Therefore, simply reducing the amount of data packed in each ciphertext would cause a large overhead in efficiency. Moreover, even with only one entry packed in one ciphertext, these schemes leak information about the data beyond a single entry. One can apply standard techniques to add data privacy (which is indeed discussed in some of the above works, but not implemented), but this may result in further decrease in efficiency, as well as other disadvantages, discussed below.

1.1 Our Contributions

In this work, we construct XSPIR, a practically efficient SPIR scheme. We follow the line of works that started with XPIR and culminated in SealPIR, MulPIR, SHECS-PIR [2,4,5,57], and add data privacy against a semi-honest client. We implement our scheme, thus providing the first implementation of a SPIR protocol in many years, and provide detailed comparisons in Sect. 4.

Crucially, we use techniques that are directly integrated with the underlying *BFV Leveled Homomorphic Encryption scheme* [10,28] (based on the ring-LWE assumption). This is in contrast with general ways to transform PIR schemes to SPIR schemes as proposed in previous works. For example, [4] discuss in their appendix how data privacy can be added on top of MulPIR, by using oblivious Pseudorandom Function (OPRF), for which the constructions are mainly based on DDH assumption [47,48]. Our technical approach enjoys the following advantages.

- Better security with low overhead: we add data privacy against a semi-honest client (namely, the client cannot learn any information beyond the retrieved entry), while paying only a small price in efficiency. Specifically, compared to the state-of-the-art PIR protocols (which leak information on data), we are about 30–40% slower in computation but marginally better in communication.
- Extended functionality: since our scheme directly builds on BFV without dependency on extra primitives, we can manipulate the BFV ciphertexts to allow retrieval of more complex functions of data entries (rather than just retrieving an individual entry). For example, if the client wants to query the summation of the cube of two entries (i.e., $x_i^3 + x_j^3$ for entries i, j), we can easily modify our scheme to achieve this functionality (relying on straightforward properties of BFV). The revelation of the circuit evaluation result will not leak any extra information about the two entries or the rest of the data.
- No new assumption: as an added benefit, our SPIR scheme does not need to rely on any additional assumption beyond the one that is used for PIR (namely Ring-LWE, that is needed for the BFV encryption scheme).

1.2 Technical Overview

We start with a brief description of how prior PIR protocols that we build on work at a high level. We first *fold the database* as a hypercube (for example, a 2-dimensional matrix), and recursively process the query for each dimension (say rows and then columns) [43,61]. Based on the BFV leveled homomorphic encryption scheme [10,28], each query is represented as a ciphertext, which would later be *obliviously expanded* to an encrypted 0/1 indicator vector [4,5]. The server then homomorphically performs the inner product between those 0/1 indicator vectors and the database, and returns the result. Note that BFV homomorphic encryption scheme allows *packing* multiple plaintexts inside one ciphertext, enabling "single instruction, multiple data" (SIMD) style homomorphic operations [11,13,32,60]. The database can be reshaped to pack more than one data entry together for better performance.

Prior PIR constructions, following the above outline, do not achieve data privacy. We identify two main causes of information leakage, and propose new techniques (directly integrated with the BFV encryption scheme) in order to overcome them efficiently.

- With *ciphertext packing* optimization, the client will get more than one data entry from the server's response ciphertext. A simple solution is to give up on full-capacity ciphertext packing to achieve data privacy. But its price is a great reduction in efficiency, because we can no longer fully utilize the "SIMD"-style operations provided by the underlying BFV scheme. This results more expensive homomorphic operations required during the server's computation. To overcome this problem, we introduce an *"oblivious masking"* procedure, which maintains the ciphertext packing feature, but can efficiently remove the undesired data entries from the packed ciphertext, without letting the server

know which data entries are kept. In addition, we integrate the oblivious masking with the PIR query procedure, so it does not introduce any extra communication cost.
– At high level, the PIR protocol works as follow: the client sends some ciphertexts to the server, the server perform some database dependent computation on the received ciphertexts and returns the result to the client. The security of the underlying homomorphic encryption scheme is protecting the information encrypted inside the original ciphertexts. But the server's result could leak information about the computation, and hence give extra information about the database other than the queried entry. We use *ciphertext sanitization* [27,31] to make sure that, even with the secret key, the client cannot learn extra information about a ciphertext other than the decrypted message.

1.3 Related Work

PIR. Private information retrieval (PIR) was introduced by Chor et al. [19], and inspired two lines of work: *information theoretic PIR* (IT-PIR) and *computational PIR* (cPIR) (we will use "PIR" to refer to cPIR by default). IT-PIR requires the database to be stored in several non-colluding servers. The client sends a query to each server and gets the result by combining the responses. IT-PIR has relative computational efficiency for each server and is information theoretic secure. However, it cannot be achieved with a single server, and the privacy relies on non-collusion of the servers, which can be problematic in practice [8,19,23,24,35]. In contrast, cPIR requires only computational security, and can be achieved with a single server. As previously discussed, there's a long line of works achieving cPIR. The computational cost for the server in all these works is quite high, which is a bottleneck for practical employment. Some of this is inherent: indeed, the server must perform at least linear (in the size of the database) amount of computation per query, or else some information will be leaked (e.g., if an entry is not touched during its computation, the server knows this is not the entry that the client is trying to retrieve). However, the existing results involve a very heavy computation beyond the size of the data (there is an additional multiplicative overhead depending on the security parameter and underlying cryptographic primitives, which is quite high). Significant progress have been made towards improving efficiency, although it remains a bottleneck.

SPIR. Symmetrically private information retrieval (SPIR) was introduced by Gertner et al. [34], who showed how transform any PIR scheme to a SPIR scheme, in the information theoretic setting. Modern cPIR schemes can also be transformed to SPIR schemes in generic ways, e.g., using an OPRF, as discussed above. However, to the best of our knowledge, the only existing implementations of SPIR proper are from over 15 years ago, and in Java [9,59]. There are some implementation of related primitives, as we discuss next.

Related Primitives. There are several works implementing database access systems with more complex queries, which include some privacy for both the client

and the server (cf., [29,40,56]); However, these schemes do not have full privacy, and allow some leakage of information about the queries.

A closely related primitive is *1-out-of-N Oblivious Transfer (OT)*. This is in fact equivalent to SPIR, but usually used in a different context where N is small, since it typically has a communication cost linear in N (while for PIR/SPIR a major goal is sublinear communication). Indeed, existing OT protocols mainly focus on constant size (say 1-out-of-2) OT, and on extending OT, namely implementing a large number of OT invocations efficiently [49,58]. The most efficient 1-out-of-N OT to date (but without implementation) is [48], where the authors construct random OT (retrieving a random location from a random database). In turn, random 1-out-of-N OT can be used at an offline stage to allow for a very efficient (but still linear) online 1-out-of-N OT.

Another relevant primitive is *Private Set Intersection (PSI)* [14,15,20,39,50]. PSI has two parties, a sender and a receiver, each holding a set of elements, who would like to privately compute the intersection of their sets. We note that most of the homomorphic encryption based PSI [14,15,20] rely on OPRF. Recently, Li, Lu, and Wu [45] used PSI for *password checkup* based on homomorphic encryption. They use a masking method by multiplying the result with a random vector to mask the redundant data entries. This approach bears some similarity with ours, but there are three problems trying to apply it to SPIR: first, it requires one extra multiplicative level, resulting in an additional overhead in both communication and computation, while our "oblivious masking" technique does not; second, this technique does not directly apply to SPIR because SPIR requires the server to send back an entry with meaningful data, so we cannot directly multiply our result by a random vector of numbers, (while in their case, they just need to send zero back as an indication); third, it doesn't prevent the server from leaking the information about its database-dependent computation due to BFV ciphertext noise, while we solve this problem by "ciphertext sanitization".

2 Preliminaries and Background

2.1 (Symmetrically) Private Information Retrieval

We focus only on single round cPIR, where the client sends a single query message and the server sends a single response message. Our protocol adheres to this form, as do other recent efficient PIR protocols.

A PIR scheme is parameterized by the database size N,[1] and consists of 3 PPT algorithms:

- $\mathsf{pp} \leftarrow \mathsf{PIR.Setup}(\lambda)$: Instantiate the protocol with security parameter λ.
- $\mathsf{q} \leftarrow \mathsf{PIR.Query}(i)$: Given an input $i \in [N]$, the client generates a query q to the server.
- $\mathsf{r} \leftarrow \mathsf{PIR.Response}(\mathsf{q}, \mathsf{DB})$: the server takes the client's query q and a database $\mathsf{DB} = (\mathsf{DB}_0, \ldots, \mathsf{DB}_{N-1})$ of N entries, and replies to the client with r.

[1] We leave the size of each element implicit as it does not affect the definition.

– $z \leftarrow \mathsf{PIR.Extract}(\mathsf{r})$: the client extracts the information from the server's reply r.

Correctness requires that, for all $i \in [N]$, for any output of the query function $\mathsf{q} \leftarrow \mathsf{PIR.Query}(i)$, for all database DB and reply $\mathsf{r} \leftarrow \mathsf{PIR.Response}(\mathsf{q}, \mathsf{DB})$ generated by the server, it has $\mathsf{PIR.Extract}(\mathsf{r}) = \mathsf{DB}_i$.

Definition 1 (Query Privacy). *We say a PIR scheme is query private if and only if for any two queries i and i', the two distributions $\mathsf{q} \leftarrow \mathsf{PIR.Query}(i)$ and $\mathsf{q} \leftarrow \mathsf{PIR.Query}(i')$ are computationally indistinguishable.*

Definition 2 (Data Privacy for Semi-Honest Client). *We say a PIR scheme is data private if and only if, for all $i \in [N]$, given query $\mathsf{q} \leftarrow \mathsf{PIR.Query}(i)$, for any two databases DB and DB' where $\mathsf{DB}_i = \mathsf{DB}'_i$, the two distributions $\mathsf{r} \leftarrow \mathsf{PIR.Response}(q, \mathsf{DB})$ and $\mathsf{r}' \leftarrow \mathsf{PIR.Response}(q, \mathsf{DB}')$ are computationally indistinguishable.*

For the rest of the paper, we use PIR to refer to a PIR scheme with query privacy only, and SPIR (or symmetric PIR) to refer to PIR with both query privacy and with data privacy for semi-honest client. In both cases, we mean computational schemes (with a single server) and one-round of communication as defined above.

We care about 2 types of complexity measures:

– Computational complexity: in particular, the server's running time for $\mathsf{PIR.Response}$ (as well as the client's running time for $\mathsf{PIR.Query}$ and $\mathsf{PIR.Extract}$, but this is typically much smaller, which typically takes only milliseconds and independent of database size).
– Communication complexity: the *upload* cost is measured by $|\mathsf{q}|$ and the *download* cost is measured by $|\mathsf{r}|$.

2.2 Homomorphic Encryption

We use homomorphic encryption scheme as a public key encryption scheme that can homomorphically evaluate arithmetic operations on messages inside ciphertexts. We can formulate it as the following 4 PPT algorithms:

– $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{HE.Setup}(1^\lambda)$: Takes security parameter λ as input and outputs public key pk, secret key sk.
– $\mathsf{ct} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}, m)$: Takes pk and a plaintext m as inputs, and outputs a ciphertext ct.
– $\mathsf{ct}' \leftarrow \mathsf{HE.Eval}(\mathsf{pk}, C, (\mathsf{ct}_1, \ldots, \mathsf{ct}_t))$: Takes pk, a circuit C and multiple input ciphertexts $(\mathsf{ct}_1, \ldots, \mathsf{ct}_t)$ and outputs a ciphertext ct'.
– $m' \leftarrow \mathsf{HE.Dec}(\mathsf{sk}, \mathsf{ct})$: Takes sk and a ciphertext ct as input and outputs a plaintext m'.

For correctness, we require that $\mathsf{HE.Dec}(\mathsf{sk}, \mathsf{HE.Enc}(\mathsf{pk}, m)) = m$ for $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{HE.Setup}(1^\lambda)$ and require $\mathsf{HE.Eval}$ to homomorphically apply the circuit C to the plaintext encrypted inside the input ciphertexts.

Definition 3 (Semantic Security). *We say a homomorphic encryption scheme is semantically secure if and only if for any two messages m and m', the two distributions* $\mathsf{ct} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}, m)$ *and* $\mathsf{ct}' \leftarrow \mathsf{HE.Enc}(\mathsf{pk}, m')$ *are computationally indistinguishable given the public key* pk.

Ciphertext Sanitization. Most homomorphic encryption scheme only cares about hiding the encrypted messages. However, the result ciphertext of the homomorphic evaluation $\mathsf{ct}' \leftarrow \mathsf{HE.Eval}(\mathsf{pk}, C, (\mathsf{ct}_1, \ldots, \mathsf{ct}_t))$ could leak some information about the circuit C, which might be harmful in some applications. One could employ a randomized sanitization proposed by Ducas and Stehlé [27] $\mathsf{HE.Sanitize}(\mathsf{pk}, \mathsf{ct})$ to achieve circuit privacy, satisfying the following:

- [Correctness] For any ciphertext ct, $\mathsf{HE.Dec}(\mathsf{sk}, \mathsf{HE.Sanitize}(\mathsf{pk}, \mathsf{ct})) = \mathsf{HE.Dec}(\mathsf{sk}, \mathsf{ct})$;
- [(Statistical) Sanitization] For any two ciphertext ct, ct' such that $\mathsf{HE.Dec}(\mathsf{sk}, \mathsf{ct}) = \mathsf{HE.Dec}(\mathsf{sk}, \mathsf{ct}')$, the two distributions after sanitizations $\mathsf{HE.Sanitize}(\mathsf{pk}, \mathsf{ct})$ and $\mathsf{HE.Sanitize}(\mathsf{pk}, \mathsf{ct}')$ are (statistically) indistinguishable given keys pk and sk.

Brakerski/Fan-Vercauteran Scheme. We use the Brakerski/Fan-Vercauteran homomorphic encryption scheme [10,28], which we refer to as the BFV scheme. Given a polynomial from the cyclotomic ring $R_t = \mathbb{Z}_t[X]/(X^D + 1)$, the BFV scheme encrypts it into a ciphertext consisting of two polynomials, where each polynomial is from a larger cyclotomic ring $R_q = \mathbb{Z}_q[X]/(X^D + 1)$ where $q > t$. We refer to t, q and D as the plaintext modulus, the ciphertext modulus, and the ring size, respectively. We require the ring dimension D to be a power of 2.

In addition to standard homomorphic operations, like addition and multiplication between a ciphertext and another ciphertext/plaintext, BFV scheme also supports *substitution* [5]. Given an odd integer k and a ciphertext ct encrypting a polynomial $p(x)$, the substitution operation $\mathsf{SUB}(\mathsf{ct}, k)$ returns a ciphertext encrypting the polynomial $p(x^k)$. For example, taking $k = 3$, an encrypted polynomial $3 + x + 5x^3$ can be substituted to be a ciphertext encrypting $3 + x^3 + 5x^9$.

3 Main Construction

In Sect. 3.1, we provide a PIR protocol, based on state-of-the-art PIR [4,5], which we will use as our starting point. Then in Sect. 3.2, we present our new techniques, and how they can be integrated with the PIR protocol to efficiently transform it to a SPIR protocol.

3.1 PIR from Homomorphic Encryption

Baseline PIR. We start from the basis for most state-of-the-art practical PIR protocols. The scheme relies on homomorphic encryption, and its simplest version is the following. Given a database $(\mathsf{DB}_0, \ldots, \mathsf{DB}_{N-1})$ of N entries, the client

initiates the query by sending N ciphertexts c_i, where the ciphertext for the desired entry encrypts 1, and all other ciphertexts encrypt 0 (that is, the ciphertexts encrypt an indicator vector). For each ciphertext, the server homomorphically multiplies it by the corresponding entry DB_i from the database, and returns the homomorphic sum of the results $\sum_{i=1}^{N} \mathsf{DB}_i \cdot c_i$, which is the encryption of the desired entry.

To achieve sublinear communication, Kushilevitz, Ostrovsky [43] and later Stern [61] proposed applying this scheme recursively: parameterized by the recursion level d, instead of viewing the database as a one-dimensional vector of length N, one can arrange it into a d-dimensional hypercube. Now each entry in the database will be indexed by a length-d vector $(i_0, \ldots, i_{d-1})$ where each index ranges from 0 to $N^{1/d}$. The retrieval process is then handled recursively, where the client sends $N^{1/d}$ ciphertexts for each level (encrypting an appropriate indicator vector), for a total of $d \cdot N^{1/d}$ ciphertexts. The server sends back one ciphertext (resulting from homomorphic operations of addition and multiplication by plaintexts).

Compressing Queries. In the above protocol, each ciphertext sent by the client encrypts a single bit, blowing up communication. To reduce communication, SealPIR [5] and MulPIR [4] instantiate the underlying homomorphic encryption scheme with the BFV scheme. Recall that in BFV, each ciphertext encrypts an element from cyclotomic ring $\mathbb{Z}_t[X]/(X^D + 1)$ where D is a power of 2, which is a degree-D polynomial with integer coefficient ranging from 0 to $t - 1$ for some large prime t. Now, instead of encrypting a single bit, a BFV ciphertext encrypts a vector consisting of the coefficients of the polynomial (i.e., D elements in $\mathbb{Z}_t$).

Specifically, to represent a query of index i, instead of sending an indicating vector of ciphertexts, SealPIR [5] first sends an encrypted monomial x^i (which can be viewed as a polynomial with coefficients being the indicating vector for i). The server then runs a procedure called *oblivious expansion* that allows it to obtain the encrypted coefficients and get the 0/1 indicator vector. Later MulPIR [4] observed that such technique works not only on a monomial x^i, but also for general polynomials, and took advantage of this for polynomials with more than one non-zero coefficients. Details of *oblivious expansion* is shown in Algorithm 1.

Packing More Information. As discussed above, the ciphertext encrypts an integer polynomial with degree D and coefficients from $\mathbb{Z}_t$. One could pack at most $D \cdot \lfloor \log t \rfloor$ bits of data inside a single ciphertext. For better efficiency, we should reshape the database so that each entry is of size $D \cdot \lfloor \log t \rfloor$ bits. For a typical choice of parameters for BFV scheme, say $D = 8192$ and $t \approx 2^{20}$ (t being a prime slightly larger than 2^{20}), that's about 20KB data per ciphertext.

Combining all these techniques, we show our PIR construction in Algorithm 2. The overall algorithm is the same as the MulPIR algorithm in [4]. We tuned the parameters in order to increase efficiency in some settings, and to allow us to add data privacy without changing to less efficient BFV parameters, as we do in the next section. Detailed performance comparisons are in Sect. 4.

Algorithm 1. Oblivious Expansion based on [4,5].
Given an input ciphertext q encrypting a polynomial $p(x)$ of degree n, return a list of n ciphertexts, encrypting the coefficients of $p(x)$.
Recall the homomorphic substituion operation: given a ciphertext ct encrypting $p(x)$ and an odd integer k, the substitution $\mathsf{SUB}(\mathsf{ct}, k)$ returns a ciphertext encrypting polynomial $p(x^k)$. We know that x^D is equal to -1 on cyclotomic ring $\mathbb{Z}_t[X]/(X^D+1)$. For polynomial $p(x) = \sum_{i=0}^{D-1} d_i \cdot x^i$, substituting it with $k = D+1$ gives $p(x^{D+1}) = \sum_{i=0}^{D-1} d_i \cdot x^{i \cdot D+i} = \sum_{i=0}^{D-1} d_i \cdot (-1)^i \cdot x^i$. Adding it back to $p(x)$ would zero out every coefficient for x_i where i is odd, and double every other coefficients. Repeatedly using similar steps for $k = D/2^j + 1$ on $p(x)$ would zero out every coefficient of x_i where i is not 0, and multiply d_0 by some power of 2. Then with some "shifting" (multiplying with some monomial x^{-2^j}), and dividing by the appropriate power of 2, given a encrypted polynomial $p(x) = \sum_{i=0}^{n-1} d_i \cdot x^i$, one can extract a vector of ciphertexts where the i^{th} ciphertext encrypts d^i.

procedure EXPAND(q, n, D) ▷ D is the ring size for the underlying BFV HE scheme
  Find $m = 2^\ell$ such that $m \geq n$
  $\mathsf{clist} \leftarrow [\mathsf{q}]$
  for $j = 0$ to $\ell - 1$ **do**
    for $k = 0$ to $2^\ell - 1$ **do**
      $c_0 \leftarrow \mathsf{clist}[k]$
      $c_1 \leftarrow x^{-2^j} \cdot c_0$ ▷ scalar multiplication
      $c'_k \leftarrow \mathsf{SUB}(c_0, D/2^j + 1) + c_0$
        ▷ SUB is the substitution in BFV HE scheme
      $c'_{k+2^j} \leftarrow \mathsf{SUB}(c_1, D/2^j + 1) + c_1$
    $\mathsf{clist} \leftarrow [c'_0, ..., c'_{2^{j+1}-1}]$
  $inverse \leftarrow m^{-1} \pmod t$ ▷ t is the plaintext modulus
  for k $= 0$ to $n - 1$ **do**
    $r_k \leftarrow \mathsf{clist}[k] \cdot inverse$
  return $(r_0, ..., r_{n-1})$

Algorithm 2. PIR Scheme (following [4])

procedure PIR.Setup(λ)
  $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{HE.Setup}(1^\lambda)$
  return $(\mathsf{pk}, \mathsf{sk})$
procedure PIR.Query($N, d, \mathsf{pk}, i = (i_0, ..., i_{d-1})$)
  Initialize polynomial $p = 0$
  for $j = 0$ to $d - 1$ **do**
    $p \leftarrow p + x^{j \cdot N^{1/d} + i_j}$
  $\mathsf{q} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}, p)$
  return (q)
procedure PIR.Response($\mathsf{DB}, N, d, \mathsf{pk}, \mathsf{q}$)
  $n \leftarrow N^{1/d}$
  $idx \leftarrow$EXPAND$(q, d \cdot n, D)$ ▷ Oblivious expansion in 1
  for $k = 0$ to $d - 1$ **do**
    $\mathsf{q}_k \leftarrow [idx[k \cdot n + 0], \ldots, idx[k \cdot n + n - 1]]$
  $\mathsf{rlist} \leftarrow [\mathsf{DB}_0, \ldots, \mathsf{DB}_{N-1}]$
  $\ell \leftarrow N/n$
  for $k = 0$ to $d - 1$ **do**
    for $i = 0$ to $\ell - 1$ **do**
      $\mathsf{r}_i \leftarrow \langle \mathsf{q}_k, [\mathsf{rlist}[0 \cdot \ell + i], \ldots, \mathsf{rlist}[(n-1) \cdot \ell + i]] \rangle$
    $\mathsf{rlist} \leftarrow [\mathsf{r}_0, \ldots, \mathsf{r}_{\ell-1}]$
    $\ell \leftarrow \ell/n$
  $\mathsf{r} \leftarrow \mathsf{rlist}[0]$
  return r
procedure PIR.Extract(sk, r)
  $z \leftarrow \mathsf{HE.Dec}(\mathsf{sk}, \mathsf{r})$
  return z

3.2 XSPIR: Adding Data Privacy

So far, we described efficient standard PIR. However, this protocol (like the ones it was based on) leaks information about the data, even to an honest client. To achieve data privacy, we need to address the following two problems:

- As previously discussed, to better utilize the plaintext space of the BFV scheme and improve efficiency, we reshaped the database so that each entry now fits in a degree-D polynomial with coefficients from $\mathbb{Z}_t$, which packs $D \cdot \lfloor \log t \rfloor$ bits of information. If the client is only allowed to learn, say, a single element from $\mathbb{Z}_t$, a simple solution would be to pack only one coefficient inside each ciphertext. However, this solution is very costly. Is it possible to pack many values (say D) inside one ciphertext for better efficiency, while the client cannot learn extra information except for only one of them?
- The server computes a *deterministic* PIR.Response procedure that depends on every part of the database. The output naturally leaks information about the server's computation and hence other parts of the database. Consider the following simple example: the client is fetching 0-th entry from a database

$\mathsf{DB} = (\mathsf{DB}_0, \mathsf{DB}_1)$ with 2 entries. After learning DB_0, the client can learn DB_1 by iterating over all possible values and simulating the server's computation. Is there a way to make the server's output ciphertext irrelevant for any part of the database other than the retrieved entry?

Instead of taking a generic approach as suggested by [4], we show how to efficiently achieve the data privacy by directly taking advantage of the underlying BFV homomorphic encryption scheme, which has many benefits as described in Sect. 1.1.

Oblivious Masking. In the previous PIR construction, one ciphertext encrypts a polynomial $p(x) = \sum_{i=0}^{D-1} d_i \cdot x^i$, where each d_i is a part of the reshaped data entry that lies in $\mathbb{Z}_t$. To address the first problem above, if the client is only allowed to learn d_k for some $k \in [D]$, we need an efficient way to obliviously remove unnecessary information (the other coefficients).

Let us start with a first attempt. To keep only the k-th part d_k of the polynomial $p(x)$, the client could send another ciphertext encrypting x^{-k}, and the server can multiply them together to get $p'(x) = x^{-k} \cdot p(x) = \sum_{i=0}^{D-1} d_i \cdot x^{i-k}$. In this case, the constant coefficient is what we are looking for. We could use a similar procedure to oblivious expansion in Algorithm 1 to extract it out.

This method brings an additional overhead as the client needs to send an additional ciphertext encrypting x^{-k}. To save this communication cost, we observe that the client is not fully utilizing the plaintext space $\mathbb{Z}_t[X]/(X^D + 1)$, as the query ciphertexts sent by the client are polynomials with 0/1 coefficients. We could embed the information k in those coefficients without introducing a new monomial, with an alternative packing technique.

First, instead of sending a new ciphertext encrypting x^{-k}, we put k into the first query ciphertext sent by the client. For example, instead of sending x^i for some index i, we send $(k+1) \cdot x^i$. After the oblivious expansion, the server can sum up the results to obtain a ciphertext encrypting a constant polynomial $(k+1)$. It requires $t > D$, which is almost always the case.

Second, instead of packing data entires $(d_0, \ldots, d_{D-1})$ into the coefficients of a polynomial, we would find a polynomial $p(x)$ such that $p(\omega_i) = d_i$ using number-theoretic transformation, where ω_i is the i-th root of unity in $\mathbb{Z}_t$, similar to the technique shown in [60]. Our goal is then to keep only the information on $p(\omega_k) = d_k$. To achieve this, we could add a random polynomial with $r(\omega_k) = 0$ to it. We first find a polynomial $q(x)$ with $q(\omega_i) = -(i+1)$. Adding to it a constant polynomial $(k+1)$ results in a new polynomial $q'(\omega_i) = k - i$. Finally, multiplying it by a random polynomial gives us what we want.

Such technique also works when the client is retrieving more than one consecutive elements in $\mathbb{Z}_t$. For example, if every data entry fits in 2 elements of $\mathbb{Z}_t$, we could find the polynomial $q(x)$ with $q(\omega_i) = \lfloor -(i/2+1) \rfloor$ instead of $-(i+1)$. And the rest of the computation would be the same.

Ciphertext Sanitization. To address the second problem and make sure that the result doesn't contain information about other parts of the database, one

way is to use the ciphertext sanitization procedure proposed by Ducas and Stehlé [27]. For efficiency, we use a simpler way of re-randomization, which is noise flooding [27,31]. Specifically, before sending back the result, the server adds an encryption of zero to it with certain amount of noise, so that the result will be statistically close to a freshly encrypted ciphertext. To achieve statistical distance of 2^{-s}, a standard smudging lemma [7] shows that it suffices to add to it an encryption of 0 with noise level $s + \log_2 D$ bits higher than the original ciphertext.

We apply all these techniques to our PIR scheme to make it into a SPIR scheme, which we call XSPIR. See Algorithm 3 for the detailed scheme.

Algorithm 3. XSPIR: Our SPIR Scheme
Blue lines are differences from the previous PIR protocol 2

procedure PIR.Setup(λ)
  $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{HE.Setup}(1^\lambda)$
  return $(\mathsf{pk}, \mathsf{sk})$
procedure PIR.Query($N, d, \mathsf{pk}, i = (i_0, ..., i_{d-1}, k)$)
  Initialize polynomial $p = 0$
  for $j = 0$ to $d - 1$ **do**
    $p \leftarrow p + (k+1) \cdot x^{j \cdot N^{1/d} + i_j}$
  $\mathsf{q} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}, p)$
  return (q)
procedure PIR.Response($\mathsf{DB}, N, d, \mathsf{pk}, \mathsf{q}$)
  $n \leftarrow N^{1/d}$
  $idx \leftarrow$EXPAND$(q, d \cdot n, D)$
  for $k = 0$ to $d - 1$ **do**
    $\mathsf{q}_k \leftarrow [idx[k \cdot n + 0], \ldots, idx[k \cdot n + n - 1]]$
  $\mathsf{rlist} \leftarrow [\mathsf{DB}_0, \ldots, \mathsf{DB}_{N-1}]$
  $\ell \leftarrow N/n$
  for $k = 0$ to $d - 1$ **do**
    for $i = 0$ to $\ell - 1$ **do**
      $\mathsf{r}_i \leftarrow \langle \mathsf{q}_k, [\mathsf{rlist}[0 \cdot \ell + i], \ldots, \mathsf{rlist}[(n-1) \cdot \ell + i]] \rangle$
    $\mathsf{rlist} \leftarrow [\mathsf{r}_0, \ldots, \mathsf{r}_{\ell-1}]$
    $\ell \leftarrow \ell / n$
  $\mathsf{r} \leftarrow \mathsf{rlist}[0]$
  $\mathsf{pt} \leftarrow (-1, \ldots, -D)$ ▷ Making a plaintext polynomial, where $\mathsf{pt}(\omega_i) = -(i+1)$
  $\mathsf{ct} \leftarrow \sum_{i=0}^{n-1} q_0[i]$ ▷ Sum of q_0 is an encrypted constant polynomial $k + 1$
  $\mathsf{ct} \leftarrow \mathsf{pt} + \mathsf{ct}$ ▷ Scalar addition
  $\mathsf{pt} \leftarrow_\$ \mathbb{Z}_t[X]/(X^D + 1)$ ▷ Uniformly sample a random polynomial
  $\mathsf{ct} \leftarrow \mathsf{pt} \cdot \mathsf{ct}$ ▷ Scalar multiplication
  $\mathsf{r} \leftarrow \mathsf{r} + \mathsf{ct}$ ▷ Homomorphic addition
  $\mathsf{r} \leftarrow \mathsf{HE.Sanitize}(\mathsf{pk}, \mathsf{r})$ ▷ Sanitize by adding an encryption of 0 with large noise
  return r
procedure PIR.Extract($\mathsf{sk}, \mathsf{r}, k, d$)
  $z \leftarrow (k+1)^{-d} \cdot \mathsf{HE.Dec}(\mathsf{sk}, \mathsf{r})$
  return z

Extended Functionality. As our scheme only relies on BFV, we can easily extend our functionality. Normally, the returned entry of PIR contains a singe data entry. However, our scheme can easily allow the returned entry to contain some computation (e.g., some complex functions) over data entries. For example, the client wants to query the summation of the cube of two entries (i.e., $x_i^3 + x_j^3$ for entries i, j), we can easily modify our scheme to achieve this functionality (relying on the properties of BFV), while maintaining full privacy (i.e., no information except for the result of the computation is revealed). However, for SPIR described in [4], this is not supported, as they rely on OPRF.

3.3 Security

The query privacy (see definition 1) follows directly from the semantic security of the underlying BFV homomorphic encryption scheme [10,28]. As the client is sending encrypted indices, and the semantic security (see definition 3) guarantees that the server cannot learn any information from the ciphertext.

Data privacy against semi-honest clients (see definition 2) is more complex. For all $k \in [N]$, given client's query $\mathsf{q} \leftarrow \mathsf{PIR.Query}(k)$, for any two databases DB and DB' where $\mathsf{DB}_k = \mathsf{DB}'_k$, consider the following two distributions $\mathsf{r} \leftarrow \mathsf{PIR.Response}(q, \mathsf{DB})$ and $\mathsf{r}' \leftarrow \mathsf{PIR.Response}(q, \mathsf{DB}')$.

Ciphertext sanitization (see 2.2 and [27]) guarantees that, for any ciphertext ct encrypting some polynomial p, the distribution $\mathsf{HE.Sanitize}(\mathsf{pk}, \mathsf{ct})$ is indistinguishable from a freshly encrypted ciphertext $\mathsf{HE.Enc}(\mathsf{pk}, p)$. Therefore both r and r' are indistinguishable from the fresh encryption of their underlying messages, respectively. We further show that r and r' encrypt messages from the same distribution. WLOG, assume that the whole database can be packed into one ciphertext and $D = N$. It is not hard to extend the argument to the general case of $N > D$. The ciphertext r is encrypting a polynomial p whose coefficients are in $\mathbb{Z}_t$ such that $p(\omega_i) = (k+1) \cdot \mathsf{DB}_i + (k-i) \cdot r_i$ where r_i is uniformly distributed over $\mathbb{Z}_t$. If $i = k$, we have $p(\omega_k) = (k+1) \cdot \mathsf{DB}_k$. Otherwise $p(\omega_i)$ is distributed uniformly at random over $\mathbb{Z}_t$ for $k \neq i \in [D]$. Similar argument works for r': r' is encrypting a polynomial p' such that $p'(\omega_i)$ is a uniform random element from $\mathbb{Z}_t$ for $i \neq k$ and $p'(\omega_k) = (k+1) \cdot \mathsf{DB}'_k = (k+1) \cdot \mathsf{DB}_k = p(\omega_k)$.

4 Implementation and Evaluation

In this section, we describe our implementation, evaluate its performance, and compare it with previous implementations. One thing to note is that, since there are no public modern SPIR implementations, we could only compare our XSPIR protocol with the state-of-the-art PIR protocols (which is not data private). We show that our performance is comparable to state-of-the-art PIR protocols while providing a stronger security guarantee.

Implementation and Experimental Setup. Our scheme is implemented on top of the SEAL homomorphic encryption library version 3.5.6 [51], with C++. We use the EXPAND algorithm from SealPIR. For SealPIR, we use the publicly

available source code [52], and run under the same environment, integrating it with our testing framework.

All experiments are running on a CPU 8th Gen Intel® Core™ i7-8550U quad-core processor, 4.2 GHz Max Turbo and 16 GB RAM, and with operating system Ubuntu 16.04. The numbers are averages of 100 trials. The SealPIR code is running with the parameters suggested by their paper and code. We implement the MulPIR on our code base with their suggested parameters. We cannot compare with SHECS-PIR [57], as their code is not publicly available. However, according to our analysis based the data provided by [57], our XSPIR performance would be comparable to theirs as well (with some variations depending on the entry size).

4.1 Parameter Choices

We have two security parameters, a computational security parameter for the underlying BFV scheme, and a statistical security parameter to apply noise flooding (necessary for ciphertext sanitization towards data privacy). We set our statistical security parameter to $s = 40$, as suggested by standard practice, and widely used in many other works [16,42,54]. According to the smudging lemma in [7], we need a noise of $s + \log D$ bits (more than the ciphertext to be sanitized) to guarantee a statistical distance of $\leq 2^{-s}$. We set our computational security parameter to $\lambda = 128$ as suggested by [3]. We set our ring size to be $D = 8192$ and therefore according to [3], we have a noise budget of 218 bits with $D = 8192, \lambda = 128$. For statistical secure parameter $s = 40$, we would then need $40 + \log_2(8192) = 53$ bits of extra noise, which gives our 165 bits of noise budget left for our entire computation. To accommodate 2.5 bytes per slot of a ciphertext, we need a prime plaintext modulus t of 21 bits, so for each level of multiplicative depth, we consume roughly 20–30 bits of noise budget. This is sufficient for a recursion level of $d = 2$, which is the most efficient choice. As for $d > 2$, the depth of homomorphic multiplication increases, and therefore results in more computational cost. Therefore, for best performance, we set $D = 8192, d = 2$ for security requirement $\lambda = 128, s = 40$.

To maximize the efficiency, we pack totally 8192×2.5bytes = 20 KB into one ciphertext. In our experiments, we select entry size = 288 bytes (this does not influence the performance, we but we select the same entry size as in previous works for better comparison). Given this entry size, we can pack at most 71 entries into one single ciphertext.

4.2 Experimental Comparisons

To evaluate how our scheme works, we run a series of microbenchmarks to measure: (1) computational cost on the server's side (2) upload communication cost (3) download communication cost. The total communication cost is measured by the sum of upload cost and download cost. Our detailed comparisons and data are recorded in Table 1.

Table 1. Entry size = 288 bytes and ring dimensions are set to 4096. In blue color is XSPIR from Algorithm 3. Although there is only one ciphertext involved in both upload and download communication. Its size varies because of the modulus switching. Other entries are PIR schemes without data privacy: SealPIR [5], MulPIR [4].

Size of database	18M	72M	288M	1.125GB
XSPIR (Server Time, ms)	1735	4921	14531	41853
SealPIR (Server Time, ms)	591	1571	6052	21675
MulPIR (Server Time, ms)	1322	3853	10785	30217
XSPIR (Upload, KB)	160	160	160	160
SealPIR (Upload, KB)	61.2	61.2	61.2	61.2
MulPIR (Upload, KB)	122	122	122	122
XSPIR (Download, KB)	73	73	73	73
SealPIR (Download, KB)	307	307	307	307
MulPIR (Download, KB)	119	119	119	119
XSPIR (Communication, KB)	233	233	233	233
SealPIR (Communication, KB)	368.2	368.2	368.2	368.2
MulPIR (Communication, KB)	241	241	241	241

As shown in the table, for all database sizes tested, our communication cost is about the same as MulPIR (with marginal advantage) and around 35% better than SealPIR, while our performance is about 40–50% worse than MulPIR and about 2–3 times worse than SealPIR. Recall that the goal in MulPIR was to obtain better communication (compared to SealPIR), at the price of worse computation. Our scheme can be viewed as going even further in that direction, but more importantly, adding a better security guarantee, for the database as well.

4.3 Comparison to 1-out-of-n OT

As mentioned in Sect. 1.3, SPIR is technically equivalent to 1-out-of-N OT, although the later one is typically used in different contexts. Accordingly, the existing open-source codes [58] for OT's focus on OT extensions, running multiple OT's at the same time. We thus can't run their library for executing a single (or a small number of) retrievals with the relatively huge database size we run experiments with, as in our XSPIR.

We next try to compare our XSPIR scheme to the state of the art 1-out-of-N OT by McQuoid et al. [48]. Since this is not implemented, we compare asymptotically. Note that in our scheme, the communication is $O(N^{1/d})$, and the server's computation is $O(N + d \cdot N^{1/d})$ homomorphic operations. In [48], they construct *random* OT, where both the query and the database are selected at random (this is typical in settings where this is used for an initial offline computation phase).

Typically, the purpose of using a 1-out-of-N random OT is to move most of the computation to an offline stage, where the random OT protocol is performed. Then, in the online stage when the client receives the actual query, it sends the difference between that and the random query used to the server. The server rotates the random data by that shift, and uses it to mask the actual database. It then sends the whole masked database to the client. The client can unmask the desired entry using the value obtained in the random OT phase. Using the random OT scheme of [48] in this way, we obtain a 1-out-of-N random OT with server time of $O(N)$ exponentiations, upload cost of $O(1)$, and download cost of $O(N)$. This gives worse communication (which is no longer sublinear!) but better computational cost than our protocol asymptotically.

Acknowledgement. This research was supported in part by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research under award number DE-SC-0001234, a grant from the Columbia-IBM center for Blockchain and Data Transparency, by LexisNexis risk solutions, and by JPMorgan Chase & Co. Any views or opinions expressed herein are solely those of the authors listed.

References

1. Abusalah, H., Alwen, J., Cohen, B., Khilko, D., Pietrzak, K., Reyzin, L.: Beyond Hellman's time-memory trade-offs with applications to proofs of space. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 357–379. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_13
2. Aguilar Melchor, C., Barrier, J., Fousse, L., Killijian, M.O.: XPIR: private information retrieval for everyone. Proc. Priv. Enhancing Technol. **2016**(2), 155–174 (2016)
3. Albrecht, M., et al.: Homomorphic Encryption Standard. In: Lauter, K., Dai, W., Laine, K. (eds.) Protecting Privacy through Homomorphic Encryption, pp. 31–62. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77287-1_2
4. Ali, A., et al.: Communication-computation trade-offs in PIR. In: 30th USENIX Security Symposium (USENIX Security 21). USENIX Association (2021). https://www.usenix.org/conference/usenixsecurity21/presentation/ali
5. Angel, S., Chen, H., Laine, K., Setty, S.T.V.: PIR with compressed queries and amortized query processing. In: 2018 IEEE Symposium on Security and Privacy, pp. 962–979. IEEE Computer Society Press (2018)
6. Angel, S., Setty, S.: Unobservable communication over fully untrusted infrastructure. In: Holz, T., Savage, S. (eds.) USENIX Security 2016: 25th USENIX Security Symposium. USENIX Association (2016)
7. Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold FHE. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 483–501. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_29
8. Beimel, A., Ishai, Y., Kushilevitz, E., Raymond, J.F.: Breaking the $O(n^{1/(2k-1)})$ barrier for information-theoretic private information retrieval. In: 43rd Annual Symposium on Foundations of Computer Science, pp. 261–270. IEEE Computer Society Press (2002)

9. Boneh, D., Bortz, A., Inguva, S., Saint-Jean, F., Feigenbaum, J.: Private information retrieval. https://crypto.stanford.edu/pir-library/
10. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 868–886. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_50
11. Brakerski, Z., Gentry, C., Halevi, S.: Packed ciphertexts in LWE-based homomorphic encryption. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 1–13. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36362-7_1
12. Cachin, C., Micali, S., Stadler, M.: Computationally private information retrieval with polylogarithmic communication. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 402–414. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_28
13. Chen, H., Dai, W., Kim, M., Song, Y.: Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019: 26th Conference on Computer and Communications Security, pp. 395–412. ACM Press (2019)
14. Chen, H., Huang, Z., Laine, K., Rindal, P.: Labeled psi from fully homomorphic encryption with malicious security. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. CCS 2018, Association for Computing Machinery (2018). https://doi.org/10.1145/3243734.3243836
15. Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. CCS 2017, Association for Computing Machinery (2017). https://doi.org/10.1145/3133956.3134061
16. Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017: 24th Conference on Computer and Communications Security, pp. 1243–1255. ACM Press (2017)
17. Cheng, R., et al.: Talek: private group messaging with hidden access patterns. Cryptology ePrint Archive, Report 2020/066 (2020). https://eprint.iacr.org/2020/066
18. Cheng, R., et al.: Talek: a private publish-subscribe protocol. In Submission (2020). https://raymondcheng.net/download/papers/talek-tr.pdf
19. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private information retrieval. In: 36th Annual Symposium on Foundations of Computer Science, pp. 41–50. IEEE Computer Society Press (1995)
20. Cong, K., et al.: Labeled PSI from homomorphic encryption with reduced computation and communication. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. CCS 2021, Association for Computing Machinery (2021). https://doi.org/10.1145/3460120.3484760
21. Costea, S., Barbu, D.M., Ghinita, G., Rughinis, R.: A comparative evaluation of private information retrieval techniques in location-based services. In: 2012 Fourth International Conference on Intelligent Networking and Collaborative Systems, pp. 618–623 (2012)
22. De Cristofaro, E., Lu, Y., Tsudik, G.: Efficient techniques for privacy-preserving sharing of sensitive information. In: McCune, J.M., et al. (eds.) Trust and Trustworthy Computing, pp. 239–253. Springer, Berlin Heidelberg, Berlin, Heidelberg (2011)

23. Demmler, D., Herzberg, A., Schneider, T.: RAID-PIR: Practical multi-server PIR. In: CCSW 2014: Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security, pp. 45–56 (2014)
24. Devet, C., Goldberg, I., Heninger, N.: Optimally robust private information retrieval. In: Kohno, T. (ed.) USENIX Security 2012: 21st USENIX Security Symposium, pp. 269–283. USENIX Association (2012)
25. Dong, C., Chen, L.: A fast single server private information retrieval protocol with low communication cost. In: Kutyłowski, M., Vaidya, J. (eds.) ESORICS 2014. LNCS, vol. 8712, pp. 380–399. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11203-9_22
26. Döttling, N., Garg, S., Ishai, Y., Malavolta, G., Mour, T., Ostrovsky, R.: Trapdoor hash functions and their applications. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 3–32. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_1
27. Ducas, L., Stehlé, D.: Sanitization of FHE ciphertexts. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9665, pp. 294–310. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49890-3_12
28. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144 (2012). http://eprint.iacr.org/2012/144
29. Fisch, B.A., et al.: Malicious-client security in blind seer: a scalable private DBMS. In: 2015 IEEE Symposium on Security and Privacy, pp. 395–410. IEEE Computer Society Press (2015)
30. Garg, S., Hajiabadi, M., Ostrovsky, R.: Efficient range-trapdoor functions and applications: rate-1 OT and more. Cryptology ePrint Archive, Report 2019/990 (2019). https://eprint.iacr.org/2019/990
31. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) 41st Annual ACM Symposium on Theory of Computing, pp. 169–178. ACM Press (2009)
32. Gentry, C., Halevi, S., Smart, N.P.: Fully homomorphic encryption with polylog overhead. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 465–482. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_28
33. Gentry, C., Ramzan, Z.: Single-database private information retrieval with constant communication rate. In: Caires, L., et al. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 803–815. Springer, Heidelberg (2005). https://doi.org/10.1007/11523468_65
34. Gertner, Y., Ishai, Y., Kushilevitz, E., Malkin, T.: Protecting data privacy in private information retrieval schemes. J. Comput. Syst. Sci. **60**(3) (2000). https://doi.org/10.1006/jcss.1999.1689
35. Goldberg, I.: Improving the robustness of private information retrieval. In: 2007 IEEE Symposium on Security and Privacy, pp. 131–148. IEEE Computer Society Press (2007)
36. Green, M., Ladd, W., Miers, I.: A protocol for privately reporting ad impressions at scale. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. CCS 2016, Association for Computing Machinery (2016). https://doi.org/10.1145/2976749.2978407
37. Groth, J., Kiayias, A., Lipmaa, H.: Multi-query computationally-private information retrieval with constant communication rate. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 107–123. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13013-7_7

38. Gupta, T., Crooks, N., Mulhern, W., Setty, S., Alvisi, L., Walfish, M.: Scalable and private media consumption with popcorn. Cryptology ePrint Archive, Report 2015/489 (2015). http://eprint.iacr.org/2015/489
39. Huberman, B.A., Franklin, M., Hogg, T.: Enhancing privacy and trust in electronic communities. In: Proceedings of the 1st ACM Conference on Electronic Commerce. EC 1999, Association for Computing Machinery (1999). https://doi.org/10.1145/336992.337012
40. Jarecki, S., Jutla, C., Krawczyk, H., Rosu, M.C., Steiner, M.: Outsourced symmetric private information retrieval. In: Proceedings of the ACM Conference on Computer and Communications Security (2013)
41. Kiayias, A., Leonardos, N., Lipmaa, H., Pavlyk, K., Tang, Q.: Optimal rate private information retrieval from homomorphic encryption. Proc. Priv. Enhancing Technol. **2015**(2), 222–243 (2015)
42. Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious PRF with applications to private set intersection. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016: 23rd Conference on Computer and Communications Security, pp. 818–829. ACM Press (2016)
43. Kushilevitz, E., Ostrovsky, R.: Replication is NOT needed: SINGLE database, computationally-private information retrieval. In: 38th Annual Symposium on Foundations of Computer Science, pp. 364–373. IEEE Computer Society Press (1997)
44. Kwon, A., Lazar, D., Devadas, S., Ford, B.: Riffle: an efficient communication system with strong anonymity. Proc. Priv. Enhancing Technol. **2016**(2), 115–134 (2016)
45. Li, J., Liu, Y., Wu, S.: Pipa: Privacy-preserving password checkup via homomorphic encryption. In: Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security (2021)
46. Lipmaa, H., Pavlyk, K.: A simpler rate-optimal CPIR protocol. In: Kiayias, A. (ed.) FC 2017. LNCS, vol. 10322, pp. 621–638. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70972-7_35
47. Mansy, D., Rindal, P.: Endemic oblivious transfer. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. CCS 2019, Association for Computing Machinery (2019). https://doi.org/10.1145/3319535.3354210
48. McQuoid, I., Rosulek, M., Roy, L.: Minimal symmetric PAKE and 1-out-of-N OT from programmable-once public functions. Cryptology ePrint Archive, Report 2020/1043 (2020). https://eprint.iacr.org/2020/1043
49. McQuoid, I., Rosulek, M., Roy, L.: Batching base oblivious transfers. Cryptology ePrint Archive, Report 2021/682 (2021). https://eprint.iacr.org/2021/682
50. Meadows, C.: A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In: 1986 IEEE Symposium on Security and Privacy, pp. 134–134 (1986)
51. Microsoft SEAL (release 3.5). Microsoft Research, Redmond, WA (2020). https://github.com/Microsoft/SEAL
52. Microsoft SealPIR. https://github.com/microsoft/SealPIR
53. Mittal, P., Olumofin, F.G., Troncoso, C., Borisov, N., Goldberg, I.: PIR-tor: scalable anonymous communication using private information retrieval. In: USENIX Security 2011: 20th USENIX Security Symposium. USENIX Association (2011)

54. Orrù, M., Orsini, E., Scholl, P.: Actively secure 1-out-of-N OT extension with application to private set intersection. In: Handschuh, H. (ed.) CT-RSA 2017. LNCS, vol. 10159, pp. 381–396. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-52153-4_22
55. Papadopoulos, S., Bakiras, S., Papadias, D.: pCloud: a distributed system for practical PIR. IEEE Trans. Dependable Secure Comput. **9**(1), 115–127 (2012)
56. Pappas, V., et al.: Blind seer: a scalable private DBMS. In: 2014 IEEE Symposium on Security and Privacy, pp. 359–374. IEEE Computer Society Press (2014)
57. Park, J., Tibouchi, M.: SHECS-PIR: Somewhat Homomorphic Encryption-Based Compact and Scalable Private Information Retrieval. In: Chen, L., Li, N., Liang, K., Schneider, S. (eds.) ESORICS 2020. LNCS, vol. 12309, pp. 86–106. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59013-0_5
58. Rindal, P.: libOTe: an efficient, portable, and easy to use Oblivious Transfer Library. https://github.com/osu-crypto/libOTe
59. Saint-Jean, F.: Java implementation of a single-database computationally symmetric private information retrieval (CSPIR) protocol. Yale University New Haven CT Department of Computer Science Technical Representative (2005)
60. Smart, N., Vercauteren, F.: Fully homomorphic SIMD operations. Cryptology ePrint Archive, Report 2011/133 (2011). http://eprint.iacr.org/2011/133
61. Stern, J.P.: A new and efficient all-or-nothing disclosure of secrets protocol. In: Ohta, K., et al. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 357–371. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-49649-1_28