

Two New Efficient PIR-Writing Protocols

Helger Lipmaa^{1,2} and Bingsheng Zhang^{1,3}

¹ Cybernetica AS, Estonia

² Tallinn University, Estonia

³ University of Tartu, Estonia

Abstract Assume that a client outsources his database to a remote storage-provider (the server), so that for privacy reasons, the client's database is encrypted by his secret key. During a PIR-writing protocol, the client updates one element of the encrypted database without revealing to the semi-honest server which element was updated and, of course, to which value. The best previous PIR-writing protocols had square-root communication complexity. In this paper, we propose two new PIR-writing protocols. The first one can be based on (say) the Damgård-Jurik additively homomorphic public-key cryptosystem, and it has (amortized) polylogarithmic communication for a limited number of updates. The second one is based on a fully-homomorphic public-key cryptosystem, a much stronger primitive, but it achieves optimal logarithmic communication.

Keywords. Cryptocomputing, fully-homomorphic encryption, PIR-writing, PrivateBDD.

1 Introduction

With the progress of network facilities, an increasing number of services are based on remote storage (also known as online disks), such as Google Doc, virtual OS and many other other web applications. Meanwhile, the clients of such services do not always trust the storage provider to keep their privacy. Therefore, any client would like to only outsource an encrypted database that only he can decrypt. On the other hand, most applications require that the storage provider should allow clients to add, retrieve, modify and delete documents of their encrypted databases. In particular, in a PIR-writing protocol (also known as private database modification, [2]), the client updates one element of the encrypted database so that the semi-honest server does not get to know which element was updated and to which value.

More precisely, assume that the unencrypted database is $f = (f_0, \dots, f_{n-1})$ of ℓ -bit elements, the client's private index is x and the private update value is y . After executing a PIR-writing protocol, the server updates the x -th element of the client's database to y . Since server's part in the PIR-writing protocol can be executed without knowing the secret key, it also possibly allows a third party to (homomorphically) modify the client's database without compromising the client's privacy, and this property makes it possible to combine PIR-writing protocols with other client-server protocols.

In a trivial PIR-writing protocol with $\Theta(n)$ communication, the server transfers the encrypted database to the client, who replaces the x -th element by an encryption of y , and then re-encrypts the database and sends the new database back. The first non-trivial solution to this problem was recently proposed in [2]. Their protocol has communication complexity $O(\sqrt{n})$ when modifying 1 bit of the database. By repeating the protocol, one has a PIR-writing protocol with the communication complexity $O(\ell\sqrt{n})$ for modifying ℓ bits.

Another—and strongly related—protocol was proposed in a yet unpublished eprint [3]. They propose a pair of amortized protocols with communication complexity $O(\sqrt{\ell^{1+\alpha} \cdot n} \cdot \text{polylog}(n))$ (where $\alpha \in (0, 1)$ is a constant) for modifying ℓ bits of the database. One protocol achieves this amortized communication complexity for arbitrary bit modifications, while the second one achieves this amortized communication complexity only when flipping a 0 bit into a 1 bit. Both square-root communication solutions use the BGN cryptosystem [1] as the underlying cryptographic primitive, so their security is based on the hardness of the Subgroup Decision assumption [1]. A drawback of the protocols of [2,3] is that they require the client to precompute the increment between the new value and the original one of the x th element, by say using an additional communication-efficient CPIR protocol to first retrieve the current value of this element. This adds complexity to their protocols, and in particular the PIR-writing protocols of [2,3] consist of more than one message. In this paper, we propose two new nontrivial PIR-writing protocols.

First New, PrivateBDD-Based PIR-Writing Protocol. The first new PIR-writing protocol is based on the cryptocomputing protocol PrivateBDD of Ishai and Paskin [9], or more precisely on an efficient variation of it as described in [11]. As described in [11], the PrivateBDD cryptocomputing protocol is based on an efficient binary decision diagram (BDD) together with an efficient $(2, 1)$ -CPIR protocol. See Sect. 2.1 for background about the BDDs (a very well known computational model) and the PrivateBDD protocol in general.

In the first new PIR-writing protocol, the server constructs a (multi-terminal) binary decision diagram (BDD, also known as branching program, [15]) for the function

$$g_i(x, y, m) := \begin{cases} y, & x = i, \\ m, & x \neq i. \end{cases}$$

The corresponding BDD has size and length $\lceil \log_2 n \rceil$. In the nonprivate version of the PIR-writing protocol, the server just sets in parallel $f_i \leftarrow g_i(x, y, f_i)$ for every $i \in \{0, \dots, n-1\}$. In the actual new PIR-writing protocol, the server uses the PrivateBDD protocol to update the encrypted version of f_i , given encryptions of x , y and a (multiple-)encryption of the old version of f_i .

Now, the output of the PrivateBDD protocol is a multiple-encryption of the actual value of the protocol, where the number of multiples is equal to the length of the BDD, that is, to $\lceil \log_2 n \rceil$ in this concrete case. Therefore, after every update, the length of every (multiple-encrypted) database element kept by the

server increases by $\lceil \log_2 n \rceil \cdot \kappa$ bits, where κ is the security parameter. More precisely, for some upper bound u on the number of updates, this PIR-writing protocol achieves *amortized* communication complexity $\Theta(u\ell \cdot \log n + u^2 \kappa \cdot \log^2 n)$. This improves upon the protocol of [3] for $u = o(\sqrt[4]{\ell^{1+\alpha} \cdot n} \cdot \text{polylog}(n))$. In addition, since the working time of the PrivateBDD is proportional to the size of the BDD, the working time of this new PIR-writing protocol is $\Theta(n \cdot \log n)$ public-key operations, where the cost of public-key operations depends on ℓ and increases after every update.

We also construct a slight modification of this protocol, where the server does not update the database elements but instead just stores all the client’s write requests. This makes the server’s computational complexity during the PIR-writing protocol very small. However, when the client requests to read some element of the database, the server executes all write requests on the original database. Therefore, this modification is useful in applications where updating is much more frequent than reading.

Importantly, this protocol does *not* use pairings. In particular, it relies on the classical DCR assumption which is by far the weakest security assumption under which a nontrivial PIR-writing protocol is known. Clearly, even in the trivial PIR-writing protocol, the database has to be encrypted by using a CPA-secure public-key cryptosystem. It is conceivable that this cryptosystem must be additively homomorphic to allow some simple cryptocomputing protocols on encrypted database. In such a case, the PrivateBDD-based PIR-writing protocol comes “free” in the sense of security, adding no extra security assumptions.

Second New, FH-Based PIR-Writing Protocol. The second new PIR-writing protocol is based on the (leveled) fully-homomorphic public-key cryptosystem proposed recently by Gentry [7]. (Equivalently, one could also use the cryptosystem from [5]) The idea behind this protocol is similar to the first new PIR-writing protocol. This time we write down a Boolean circuit for $g_i(x, y, f_i)$, which happens to have exactly the same size as the corresponding BDD but has depth $\approx \log_2 \log_2 n$. We now use the fully-homomorphic cryptosystem to evaluate in parallel n copies of this circuit for $i \in \{0, \dots, n-1\}$. Importantly, (1) this circuit only has multiplicative depth $\log \log n$ (as compared to the length $\log n$ of the corresponding BDD), and (2) this process does *not* increase the size of the server’s database. The actual FH-based PIR-writing protocol has communication complexity $O((\log n + \ell) \cdot \kappa)$, and computation complexity of $O(n \cdot \log n + \ell n)$ evaluations of the fully-homomorphic cryptosystem.

However, since every multiplication (or Boolean AND) increases dramatically the noise used in encryption, some care has to be taken to bootstrap the stored database values, see [7]. Since the multiplicative depth of n parallel applications of the circuit for g is $\lceil \log_2 \lceil \log_2 n \rceil \rceil$ (which is never larger than 5 in practice), we can just assume that the security parameter is big enough to accommodate the evaluation of circuits of multiplicative depth $\Theta(\log \log n)$. However, a bootstrapping should be done at the end of the PIR-writing protocol. See [7,6] for more discussion.

Scheme	Communication Complexity	Computation Complexity	Security Assumption	Increase Database Size
Trivial	$n(\ell + \kappa)$	$O(\ell n)$	CPA-security of underlying cryptosystem	None
[2]	$O(\ell\sqrt{n})$	$O(\ell n)$	subgroup decision [1] + polylog-communication CPIR [8,10]	None
[3]	$\sqrt{\ell^{1+\alpha} \cdot n} \cdot \text{polylog}(n)$	$O(n \cdot \text{polylog}(n))$	subgroup decision [1] + polylog-communication CPIR [8,10]	None
Sect. 3	$\Theta(u\ell \cdot \log n + u^2\kappa \cdot \log^2 n)$	$O(n \cdot \log n)$	DCR assumption	Increased by $\kappa \log n$
Sect. 3.1	$\Theta(u\ell \cdot \log n + u^2\kappa \cdot \log^2 n)$	Trivial	DCR assumption	None
Sect. 4	$O(\kappa \log n + \kappa\ell)$	$O(n \cdot \log n + \ell n)$	CPA-security of fully-homomorphic cryptosystem	None

Table 1. Comparison of previous PIR-writing protocols and our two new protocols. In the case of the protocol from Sect. 3 (and Sect. 3), u is the number of updates, and we give amortized communication over the first u updates. Note that the meaning of the unit computation depends on the protocol

In particular, if we use *leveled* fully-homomorphic cryptosystem, then the length of the public key is linear in U , where U is the maximal number of imagined updates. Since this basically means that one element of the public key has to be transferred to the server per every update, it does not increase the amortized communication cost significantly. Moreover, if we assume that the fully-homomorphic cryptosystem is secure against key-dependent message (KDM) attacks, then it is fully-homomorphic (and not leveled fully-homomorphic) and we can just use a constant-in- U -size public key. Since the security of the cryptosystems from [7,5] has not been well-studied with or without KDM attacks, we will not mention the cost of maintaining and transferring the public key anymore.

Comparison. In Table 1, we compare the trivial solution, two previously known sublinear-communication protocols, and the two new protocols. Note that in the protocol from Sect. 3, the cost of a single public-key operation depends on ℓ , and on the number of the update.

2 Preliminaries

Notation. Within this paper, κ is always the security parameter. The client has outsourced a database $f = (f_0, \dots, f_{n-1})$ to the server. Client's private inputs are an $m = \lceil \log_2 n \rceil$ -bit index $x = (x_0, \dots, x_{m-1})$ and an ℓ -bit value

$y = (y_0, \dots, y_{\ell-1})$. All logarithms are taken on basis 2. For a set (or possibly a probabilistic algorithm) S , $x \leftarrow S$ means a uniformly random assignment of an element from S to x . Linearity and polylogarithmicity is usually measured with respect to n , while in security proofs polynomiality and negligibility is measured with respect to \ll .

2.1 PrivateBDD Protocol

Length-Flexible Homomorphic Public-Key Cryptosystems. Let $\mathsf{P} = (\mathsf{G}, \mathsf{E}, \mathsf{D})$ be a length-flexible additively-homomorphic public-key cryptosystem [4], where G is a randomized key generation algorithm, E is a randomized encryption algorithm and D is a decryption algorithm. Here, both E and D receive an additional length parameter ℓ , so that $\mathsf{E}_{\mathsf{pk}}(\ell, \cdot)$ encrypts plaintexts from some set $\{0, 1\}^{\leq \ell}$. In the case of the DJ01 cryptosystem from [4], for every integer $\ell > 0$, $\mathsf{E}_{\mathsf{pk}}(\ell, \cdot)$ is a valid plaintext of $\mathsf{E}_{\mathsf{pk}}(\lceil \ell/\kappa \rceil \cdot \kappa + \kappa, \cdot)$, and therefore one can multiple-encrypt messages as say in

$$C \leftarrow \mathsf{E}_{\mathsf{pk}}(\ell + 2\kappa, \mathsf{E}_{\mathsf{pk}}(\ell + \kappa, \mathsf{E}_{\mathsf{pk}}(\ell, M))) ,$$

and then recover M by multiple-decrypting,

$$M \leftarrow \mathsf{D}_{\mathsf{sk}}(\ell + 2\kappa, \mathsf{D}_{\mathsf{sk}}(\ell + \kappa, \mathsf{D}_{\mathsf{sk}}(\ell, C))) .$$

If N is the public key of the DJ01 cryptosystem, then $2^\ell < N$. Additionally, in any length-flexible additively-homomorphic cryptosystem, $\mathsf{E}_{\mathsf{pk}}(\ell, M_1) \cdot \mathsf{E}_{\mathsf{pk}}(\ell, M_2) = \mathsf{E}_{\mathsf{pk}}(\ell, M_1 + M_2)$, where the addition is modulo the public key N . We will explicitly need the existence of a compression function C that, given pk , ℓ' and ℓ for $\ell' \geq \ell$, and $\mathsf{E}_{\mathsf{pk}}(\ell', M)$ for $M \in \{0, 1\}^\ell$, returns $\mathsf{E}_{\mathsf{pk}}(\ell, M) \in \{0, 1\}^{\lceil \ell/\kappa \rceil \cdot \kappa + \kappa}$.

In the CPA (*chosen-plaintext attack*) game, the challenger first generates a random $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{G}(1^\kappa)$, and sends pk to the attacker. Attacker chooses two messages (M_0, M_1) (such that $|M_0| = |M_1|$) and a length parameter ℓ , and sends them to the challenger. Challenger picks a random bit b , and sends a ciphertext $\mathsf{E}_{\mathsf{pk}}(\ell, M_b)$ to attacker. Attacker outputs a bit b' , and wins if $b = b'$.

In the LFCPA (*length-flexible chosen-plaintext attack*) game [10], the challenger first generates a random $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{G}(1^\kappa)$, and sends pk to the attacker. Attacker chooses a polynomial number of message pairs (M_{j0}, M_{j1}) (such that $|M_{j0}| = |M_{j1}|$) and length parameters ℓ_j , and sends them to the challenger. Challenger picks a random bit b , and sends all ciphertexts $\mathsf{E}_{\mathsf{pk}}(\ell_j, M_{jb})$ to attacker. Attacker outputs a bit b' , and wins if $b = b'$. Because of the existence of the compress function, LFCPA security follows from the CPA security [11]. Thus, the DJ01 cryptosystem [4] is LFCPA-secure under the Decisional Composite Residuosity Assumption.

Cryptocomputing. Let m and ℓ be public parameters, and let \mathcal{F} a class of functions $\{0, 1\}^m \rightarrow \{0, 1\}^\ell$. In a cryptocomputing protocol for \mathcal{F} between a

client and a server, the client has an input $x \in \{0, 1\}^m$ and the server has an input $f \in \mathcal{F}$. The client obtains $f(x)$. Every cryptocomputing protocol $\Gamma = (\mathbf{Q}, \mathbf{R}, \mathbf{A})$ has two messages, where the client sends $\mathbf{Q}(\ell, x)$ to the server, the server replies with $\mathbf{R} \leftarrow \mathbf{R}(\ell, f, \mathbf{Q})$, and then finally the stateful client recovers f_x by computing $\mathbf{A}(\ell, x, \mathbf{R})$. Here, \mathbf{Q} , \mathbf{R} and \mathbf{A} are (probabilistic) polynomial-time algorithms.

Client-Privacy of Cryptocomputing Protocols. Let $\Gamma = (\mathbf{Q}, \mathbf{R}, \mathbf{A})$ be a 2-message cryptocomputing protocol. Within this work we use the convention of many previous papers to only require (semisimulatable) privacy in the malicious model. In particular, client’s privacy is guaranteed in the sense of indistinguishability (CPA-security), That is, for the privacy of the client, no malicious nonuniform probabilistic polynomial-time server should be able to distinguish, with non-negligible probability, between the distributions $\mathbf{Q}(\ell, x_0)$ and $\mathbf{Q}(\ell, x_1)$ that correspond to any two of client’s inputs x_0 and x_1 that are chosen by herself. Within this paper, we are not interested in server-privacy.

Computationally-Private Information Retrieval. A two-message 1-out-of- n computationally-private information retrieval protocol, $(n, 1)$ -CPIR, is a special type of cryptocomputing protocol. In a $(n, 1)$ -CPIR protocol for ℓ -bit strings, the client has an index $x \in \{0, \dots, n-1\}$ and the server has a database $f = (f_0, \dots, f_{n-1})$ with $f_i \in \{0, 1\}^\ell$. The client obtains f_x . An $(n, 1)$ -CPIR protocol $\Gamma = (\mathbf{Q}, \mathbf{R}, \mathbf{A}, \mathbf{C})$ is *BDD-friendly* if it satisfies the next four assumptions:

1. Γ has two messages, a query $\mathbf{Q}(\ell, x)$ from the client and a reply $\mathbf{R}(\ell, f, \mathbf{Q})$ from the server, such that the stateful client can recover f_x by computing $\mathbf{A}(\ell, x, \mathbf{R}(\ell, f, \mathbf{Q}))$. Note that $\mathbf{A}(\ell, x, \mathbf{R}(\ell, f, \mathbf{Q}(\ell, x))) = f_x$.
2. Γ is uniform in ℓ , that is, it can be easily modified to work on other values of ℓ .
3. $|\mathbf{Q}(\ell, \cdot)|, |\mathbf{R}(\ell, \cdot, \cdot)| \leq \ell + \Theta(\kappa)$ (with possibly $\mathbf{Q}(\ell, \cdot)$ being even shorter).
4. The compress function \mathbf{C} maps $\mathbf{Q}(\ell', x)$ to $\mathbf{Q}(\ell, x)$ for any $\ell' \geq \ell$ and any x .

Here \mathbf{Q} , \mathbf{R} , \mathbf{A} and \mathbf{C} are (probabilistic) polynomial-time algorithms. The only known BDD-friendly $(2, 1)$ -CPIR was proposed by Lipmaa in [10], see [11] for a compact description. Importantly for us, in Lipmaa’s $(2, 1)$ -CPIR protocol, $\mathbf{Q}(\ell, x)$ consists of a public key and an additively homomorphic encryption of x under this key.

Any $(n, 1)$ -CPIR protocol Γ must be client-private, that is, CPA-secure. Lipmaa’s $(2, 1)$ -CPIR protocol [10], when based on the DJ01 cryptosystem [4], is CPA-secure and thus LFCPA-secure (which is defined in the same way as LFCPA-security for public-key cryptosystems) under the Decisional Composite Residuosity Assumption.

Binary Decision Diagrams. A (multi-terminal) *binary decision diagram* (BDD, also known as a branching program, [15]) is a fanout-2 directed acyclic graph $(\mathcal{V}, \mathcal{E})$, where the non-terminal nodes are labeled by variables from some

variable set $\{x_0, \dots, x_{m-1}\}$, the sinks are labeled by ℓ -bit strings and the two outgoing edges of every internal node are respectively labeled by 0 and 1. A (multi-terminal) BDD computes some function $f : \{0, 1\}^m \rightarrow \{0, 1\}^\ell$. Every assignment of the variables selects one path from the source to some sink as follows. The path starts from the source. If the current version of path does not end at a sink, test the variable at the endpoint of the path. Select one of the outgoing edges depending on the value of this variable, and append this edge and its endpoint to the path. If the path ends at a sink, return the label of this sink as the value of the corresponding source. The BDD's value is then equal to the source value. For a BDD P , let $\text{len}(P)$ be its length (that is, the length of its longest path), and let $\text{size}(P)$ be its size (that is, the number of non-terminal nodes).

PrivateBDD Protocol. In [9], Ishai and Paskin proposed a new cryptocomputing method (PrivateBDD) that uses a BDD-representation of the target function in conjunction with a communication-efficient strong oblivious transfer. In [11], the authors noted that the strong oblivious transfer protocol can be replaced by a BDD-friendly (2, 1)-CPIR protocol. We now briefly recall the main properties of PrivateBDD, as instantiated by Lipmaa's (2, 1)-CPIR from [10]. See [11] for the full details of the PrivateBDD protocol.

Theorem 1. *Assume that the Decisional Composite Residuosity Assumption is true. Let \mathcal{F} be a set of functions $f : \{0, 1\}^m \rightarrow \{0, 1\}^\ell$, and for any $f \in \mathcal{F}$ let P_f be some (multi-terminal) BDD with ℓ -bit sink labels that computes f . Let $\text{len}(\mathcal{F}) := \max_{f \in \mathcal{F}} \text{len}(f)$. Then \mathcal{F} has a CPA-secure cryptocomputing protocol with communication upperbounded by $\kappa + m \cdot (\ell + (\text{len}(\mathcal{F}) + 2) \cdot \kappa)$, and server's online computation dominated by $\text{size}(f)$ public-key operations.*

Briefly, client's inputs to the PrivateBDD (when instantiated by Lipmaa's (2, 1)-CPIR from [10]) are encrypted bitwise by using a length-flexible additively homomorphic public-key cryptosystem like DJ01 [4]. Moreover, let V be any internal node of the BDD such that the longest path between V and any sink has length $\text{len}(V) > 0$. Let V_0 and V_1 be the successors of V by the 0-edge and 1-edge, correspondingly. Then V 's value $\text{val}[V]$ as recursively computed by the PrivateBDD protocol is

$$\text{R}(\ell + (\text{len}(V) - 1)\kappa, \text{Q}(\ell + (\text{len}(V) - 1)\kappa, x_j), (\text{val}[V_0], \text{val}[V_1])) ,$$

where x_j is V 's label, and $\text{val}[V_i]$ is the already known value of the node V_i . Moreover, sink values are equal to their labels. Therefore, $\text{val}[V]$ is equal to an encryption of $\text{val}[V_{x_j}]$. Inductively, $\text{val}[V]$ is equal to an $\text{len}(V)$ -times encryption of some sink value, and $|\text{val}[V]| \approx (\text{len}(V) + 1)\kappa$. In particular, server's message in the PrivateBDD protocol is equal to a $\text{len}(P_f)$ -times encryption of some sink value, and this sink value by itself is the output of the PrivateBDD protocol. See [11] for more details.

2.2 Fully-Homomorphic Cryptosystem

In this subsection, we give a brief description of the recent fully-homomorphic cryptosystem by Gentry [7]. We omit all the details of Gentry’s cryptosystem that are not relevant to the current paper. In particular, all given details are also true for the more recent cryptosystem of van Dijk, Gentry, Halevi and Vaikuntanathan [5].

Let $\mathcal{P} = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ be the initial version of Gentry’s cryptosystem [7]. The length ℓ of plaintext space \mathcal{P} is fixed. Similarly to the previous subsection, \mathcal{G} is a randomized key generation algorithm such that $(\mathbf{sk}, \mathbf{pk}) \leftarrow \mathcal{G}(1^\kappa)$; \mathcal{E} is randomized encryption algorithm such that for any $M \in \mathcal{P}$, $C \leftarrow \mathcal{E}_{\mathbf{pk}}(M)$; \mathcal{D} is decryption algorithm such that for any $C \in \mathcal{C}$, where \mathcal{C} is ciphertext space, $M \leftarrow \mathcal{D}_{\mathbf{sk}}(C)$.

While encrypting, Gentry’s cryptosystem masks the plaintext in particular with an additive noise R from some “small” set \mathcal{R}_0 , and correct decrypting is guaranteed when the noise belongs to some “large” set \mathcal{R}_1 , $\mathcal{R}_0 \subset \mathcal{R}_1$. Since the noise is additive, one has

$$\mathcal{E}_{\mathbf{pk}}(M_1; R_1) + \mathcal{E}_{\mathbf{pk}}(M_2; R_2) = \mathcal{E}_{\mathbf{pk}}(M_1 + M_2; R_1 + R_2)$$

and

$$\mathcal{E}_{\mathbf{pk}}(M_1; R_1) \cdot \mathcal{E}_{\mathbf{pk}}(M_2; R_2) = \mathcal{E}_{\mathbf{pk}}(M_1 \cdot M_2; R_1 \cdot R_2) .$$

Here, the addition of plaintexts is modular, while the noise increases unboundedly. Thus, one can evaluate arbitrary $+/\cdot$ (or alternatively, \oplus/\wedge) circuits of only some bounded length before the noise has increased to the level where correct decryption is not anymore possible. Therefore, this simple version of Gentry’s cryptosystem is only somewhat homomorphic [7], that is, homomorphic for small-depth circuits.

However, as shown in [7], the somewhat homomorphic version of Gentry’s cryptosystem is sufficient to homomorphically evaluate its own decryption circuit augmented with basic Boolean operations. Hence, one can strengthen the somewhat homomorphic version with a bootstrapping step. Assume that the plaintexts have been encrypted by using some public key \mathbf{pk}_1 . Now, just before the circuit depth has reached the level where decryption becomes incorrect, one encrypts the ciphertexts $\mathcal{E}_{\mathbf{pk}_1}(\cdot)$ by using a different public key \mathbf{pk}_2 , and then homomorphically decrypts the results, obtaining new encryptions of the same plaintexts but under the new key \mathbf{pk}_2 and with decreased noise. This step is called *bootstrapping*. After that, one can homomorphically execute another few levels of the circuit, until one needs to bootstrap again.

Thus, given the somewhat homomorphic cryptosystem, Gentry constructed a leveled fully-homomorphic cryptosystem that enables one to homomorphically evaluate circuits of any *bounded* and *a priori fixed* depth d . In such a *leveled fully-homomorphic cryptosystem*, the public-key size is linear in d , and in particular it includes encryptions of all bits of \mathbf{sk}_i under the public key \mathbf{pk}_{i+1} , for linear-in- d needed public/secret key pairs $(\mathbf{sk}_i, \mathbf{pk}_i)$.

To overcome this restriction, one can assume that Gentry’s cryptosystem is secure against key-dependent message (KDM) attacks. In this case, one can,

given a *single* valid key pair $(\text{sk}, \text{pk}) \leftarrow \mathbf{G}(1^\kappa)$, circularly encrypt the bits of sk by using pk . Therefore, after every few levels of the circuit, one can use the same public key pk to bootstrap the circuit. In particular, Gentry showed [7,6] that his leveled fully-homomorphic cryptosystem is fully-homomorphic under the random oracle assumption. Therefore, within this paper, we will assume that Gentry’s cryptosystem is fully-homomorphic (and not leveled fully-homomorphic).

Finally, since we only need to encrypt Boolean plaintexts 0 and 1, other details of Gentry’s cryptosystem—for example, the fact that it is lattice-based—are not important for our purposes. We refer an interested reader to [7,6] for many further details. In particular, we will only assume that Gentry’s fully-homomorphic cryptosystem is CPA-secure (and KDM-secure). The underlying assumptions that are needed for CPA-security (and KDM-security) can again be found from [7].

2.3 PIR-Writing

Assume that the client has outsourced his database to the server. To protect his privacy, the database is encrypted by using client’s public key. In a PIR-writing protocol (also known as a private database modification protocol, [2]), the client updates a single element of the database so that the server does not know which element was changed. More precisely, the database $f = (f_0, \dots, f_{n-1})$ has n elements $f_i \in \{0, 1\}^\ell$. The client has private inputs (sk, x, y) , where sk is his secret key, $x \in \{0, \dots, n-1\}$ is the element to be changed, and $y \in \{0, 1\}^\ell$ is its new version. The server has an encrypted version $(c_0, \dots, c_{n-1}) = (\mathbf{E}_{\text{pk}}(f_0), \dots, \mathbf{E}_{\text{pk}}(f_{n-1}))$ of the database and a copy of client’s public key pk . Ideally, the protocol consists of only a single message, from the client to the server. The client has no private output, while server obtains a new encrypted database $c' = (c'_0, \dots, c'_{n-1})$, such that c'_i and c_i decrypt to the same value if $i \neq x$, while c'_x decrypts to y .

The privacy definition of a PIR-writing protocol is formalized by the following PIR-writing game. Let A be a semi-honest probabilistic-polynomial time adversary (that is, the server), and let C be the challenger. The game consists of the following steps:

1. The challenger C generates a pair of keys $(\text{sk}, \text{pk}) \leftarrow \mathbf{G}(1^\kappa)$, and sends pk to A .
2. A picks and sends to C a database $f = (f_0, \dots, f_{n-1})$ of n elements with length ℓ .
3. C encrypts the database with pk and sends it back to A .
4. (Challenge phase:) A picks and sends to C two index and value pairs $(x_0^*, y_0^*), (x_1^*, y_1^*)$. C picks $b_c \leftarrow \{0, 1\}$, and executes the PIR-writing protocol with input (x_b^*, y_b^*) , with A playing the role of the server.
5. A outputs her guess $b_c^* \in \{0, 1\}$ for b_c .

Note that here we do not have a query phase, since in our one-message protocols the malicious server can just play the PIR-writing part with herself.

(Recall that the database she has is encrypted by using client’s public key.) The situation is different in say [2] where the client of the PIR-writing protocol had to know the current value of the modified database element.

Definition 1 (Client-privacy of PIR-writing). *Let the adversary’s advantage in the previous game be*

$$\text{Adv}_A(1^\kappa) := \left| \Pr[b_c^* = b_c] - \frac{1}{2} \right| .$$

We say that a PIR-writing protocol is client-private, if for all probabilistic-polynomial time adversaries A , $\text{Adv}_A(1^\kappa)$ is a negligible function (in κ).

Previous PIR-Writing Protocols. In a trivial (two-message) linear-communication protocol, the server sends the encrypted database back to the client, who updates the x th element, re-encrypts other elements, and sends the new encrypted database back to the server. Another linear-communication PIR-writing protocol can be based on an arbitrary additively homomorphic public-key cryptosystem as follows. The client and the server first execute an $(n, 1)$ -CPIR protocol, so that the client obtains the current value of f_x . Then client forwards to the server n ciphertexts c_j , where c_x decrypts to $y - f_x$ and other c_j -s decrypt to 0. The server multiplies the encryptions of f_j with the ciphertexts c_j , this clearly correctly updates the database.

The first sublinear-communication PIR-writing protocol was proposed in [2]. Essentially, it uses the bilinear-pairing based cryptosystem of [1] to send $2 \cdot \sqrt{n}$ ciphertexts c'_j and c''_j —such that the decryption of c_j is equal to the product of decryptions of c'_j and c''_j —instead of n ciphertexts as in the previous protocol. Thus, this protocol has communication complexity $O(\sqrt{n})$ to modify one bit of a database. Clearly, by repeating the protocol, one will have a PIR-writing protocol with communication complexity $O(\ell \cdot \sqrt{n})$ for modifying ℓ bits.

A way to decrease the communication complexity for larger ℓ was proposed in an unpublished eprint [3]. The solution consists of a pair of amortized protocols that have communication complexity $O(\sqrt{\ell^{1+\alpha} \cdot n} \cdot \text{polylog}(n))$ (where $\alpha \in (0, 1)$ is a constant) for modifying ℓ bits of the database. The idea is to encode an n -bit database as a Dn -bit “virtual database” shared on M different servers by using unbalanced lossless expander graphs. One of their protocol achieves the claimed amortized communication complexity for any arbitrary bit modifications, while the other one achieves the same amortized communication complexity only when flipping a 0 bit into a 1 bit. All protocols from [2,3] use the BGN cryptosystem [1] as cryptographic primitive, so their security is based on the hardness of subgroup decision problem [1].

A drawback of the protocols from [2,3] is that the client has to know the current value of f_x before applying their protocols. Thus, the client has to use a communication-efficient CPIR protocol [10,8] first to retrieve f_x . This increases both the computational and communication complexity. Most importantly, this means that the protocols of [2,3] have more than one message.

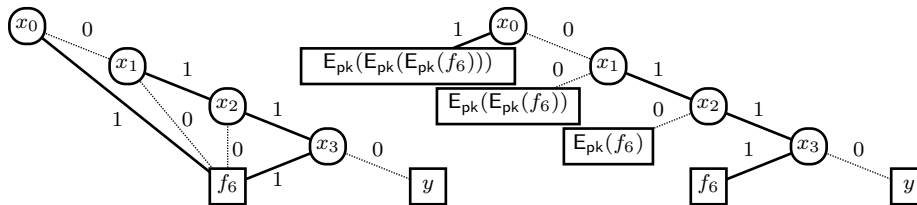


Figure 1. Left: the BDD $P_6(x, y, f_6)$ that returns y if $x = 6$, and returns f_6 otherwise. The BDD outputs y only if $x_0 = 0, x_1 = 1, x_2 = 1$ and $x_3 = 0$. Right: the BDD \tilde{P}_6 that is actually used in the new PIR-writing protocol. Here $n = 16$ and $m = 4$.

Finally, in [13], the authors studied the same problem in a different, information-theoretic setting with multiple databases.

3 New PrivateBDD-Based PIR-Writing Protocol

The first new PIR-writing protocol is based on the cryptocomputing protocol PrivateBDD of Ishai and Paskin [9]. Thus, it is based on an $(2, 1)$ -CPIR protocol of Lipmaa [10] and in particular on a length-flexible additively-homomorphic cryptosystem $\mathbf{P} = (\mathbf{G}, \mathbf{E}, \mathbf{D})$. Briefly, in this protocol the server constructs a binary decision diagram (BDD) $P_i(x, y, m)$ for the function

$$g_i(x, y, y') := \begin{cases} y & , \quad x = i & , \\ y' & , \quad x \neq i & . \end{cases}$$

(Here, we assume that n is implicitly fixed.) Denote $m := \lceil \log_2 n \rceil$. Recall that $x = (x_0, \dots, x_{m-1})$ and $y = (y_0, \dots, y_{\ell-1})$.

Lemma 1. *Let i be known and fixed. Then the functionality $g_i(x, y, y')$ can be implemented by a BDD $P_i(x, y, y')$ of size and length $m = (1 + o(1)) \log_2 n$.*

Proof. Formally, $P_i(x, y, y')$ consists of m nodes $V_i, i \in \{0, \dots, m-1\}$, such that the i th node is labeled by x_i . It also has two sinks labeled by y and y' . Moreover, there is an x_i -edge from V_i to V_{i+1} for $i < m-1$, and an x_i -edge from V_{m-1} to the sink labeled by y . Finally, there is an $(1 - x_i)$ -edge from every V_i to the sink labeled by y' . \square

As an example, $P_6(x, y, f_6)$ is depicted by Fig. 1, left.

In the nonprivate version of the resulting PIR-writing protocol, the server just sets in parallel $f_i \leftarrow g_i(x, y, f_i)$ for every $i \in \{0, \dots, n-1\}$. (She can do it by using the BDD P_i , or by any other means.)

In the new PrivateBDD-based PIR-writing protocol, the server has to use a slightly different BDD \tilde{P}_i , as depicted by Fig. 1, right. Namely, due to the design of the PrivateBDD protocol, the siblings of any internal node have to have the

same length. This is necessary to protect client's privacy. See Sect. 2.1 and [11] for more discussion. Since the value of the 0-sibling of the source on Fig. 1 is of form $E_{pk}(E_{pk}(E_{pk}(\cdot)))$, the easiest way to achieve the same length for the 1-sibling is to use $E_{pk}(E_{pk}(E_{pk}(f_6)))$ as the value of its 1-sibling. The formal description of \tilde{P}_i follows straightforwardly, and we will omit it.

Now, the server uses the PrivateBDD protocol with \tilde{P}_i to update the encrypted version of f_i , given encryptions of x , y and an encryption of the old version of f_i . Thus, instead of sending the output back to the client, as in the original PrivateBDD protocol, the server uses it to update a value that is privately held by herself.

A formal protocol description of the new PrivateBDD-based PIR-writing protocol follows. Note that since we use a length-flexible cryptosystem, we can have $\ell > \kappa$.

PrivateBDD-Based New PIR-Writing Protocol

Common inputs: Database size n , $m \leftarrow \lceil \log_2 n \rceil$, element length ℓ .

Client's inputs: Secret key sk , $x = (x_0, \dots, x_{m-1})$, $y \in \{0, 1\}^\ell$.

Server's inputs: Public key pk , $c = (c_0, \dots, c_{n-1})$, where c_j is a multiple-encryption of f_j .

Server's private output: Updated database $c' = (c'_0, \dots, c'_{n-1})$, where c'_j is a multiple-encryption of updated f'_j .

1. Client sends to the server $E_{pk}(x_i)$, for $i \in \{0, m-1\}$, and $E_{pk}(y)$.
2. The server does:
 - (a) For $i \in \{0, \dots, n-1\}$, the server executes PrivateBDD by using $\tilde{P}_i(x, y, c_i)$, and sets c'_i to be equal to its output.
 - (b) The server stores $c = (c'_0, \dots, c'_{n-1})$ as the new database with elements having length $\ell' \leftarrow \ell + \kappa \cdot \lceil \log_2 n \rceil$.

Security. We claim the client-privacy only in the special case when the PrivateBDD protocol is based on Lipmaa's (2, 1)-CPIR protocol from [10]. Obviously, this result can be generalized if other communication-efficient (2, 1)-CPIR protocols were to be constructed.

Theorem 2. 1) *The PrivateBDD-based PIR-writing protocol is correct.*
2) *If the Decisional Composite Residuosity assumption [14] holds, then the PrivateBDD-based PIR-writing protocol is client-private in the presence of a computationally-bounded malicious server.*

Proof (Sketch.) 1) The correctness of the PrivateBDD-based PIR-writing protocol follows from the correctness of the PrivateBDD protocol by Ishai and Paskin [9], and from the correctness of the BDDs \tilde{P}_i .

2) The DJ01 cryptosystem used in the this PIR-writing is known [4] to be CPA-secure under Decisional Composite Residuosity Assumption [14]. Because

Lipmaa's $(2, 1)$ -CPIR is BDD-friendly and CPA-secure, the CPA-security of the PrivateBDD follows from a standard hybrid argument. \square

Efficiency. Assume that the database elements have length ℓ . Recall that we are using the Damgård-Jurik cryptosystem [4]. Then before the first run of the PIR-writing protocol, the server has a database of once-encrypted elements that have size $(s + 1)\kappa$, where $s = \lceil \ell/\kappa \rceil$. The output of the PrivateBDD protocol, as modified by Lipmaa [11], is a multiple encryption of the actual value of corresponding BDD sink value, where the number of multiples is equal to the depth of the BDD, that is, to $m = \lceil \log_2 n \rceil$ in this concrete case. Therefore, after every update, the (multiple-encrypted) database elements kept by the server increases by $m \cdot \kappa$ bits.

More precisely, the first run of the PIR-writing protocol has communication complexity

$$\leq (m + 1) \cdot (\text{len}(P)\kappa + \ell) = (m^2 + m) \cdot \kappa + (m + 1)\ell .$$

The new database has elements of size

$$(s + m + 1) \cdot \kappa \leq (m + 1) \cdot \kappa + \ell$$

and thus the next update protocol has communication

$$(m^2 + m) \cdot \kappa + (m + 1) \cdot (m\kappa + \kappa + \ell) .$$

Analogously, the j th update protocol has communication

$$(m^2 + m) \cdot \kappa + (m + 1)(jm\kappa + j\kappa + \ell) ,$$

and thus the first u protocols have total communication

$$u(m^2 + m) \cdot \kappa + (m + 1) \sum_{j=0}^{u-1} (jm\kappa + j\kappa + \ell) = O(mul + m^2u^2\kappa) .$$

Since the working time of the PrivateBDD protocol is proportional to the size of the BDD, the computation of the new PrivateBDD-based PIR-writing protocol is dominated by $n \cdot (2m - 1)$ public-key operations, where the cost of public-key operations increases after every update. Here, $m - 1$ public-key operations come from the need to multiple-encrypt every database element f_i , so that it would have correct length to be on the correct layer of \tilde{P}_i . However, since we do not care about server's privacy at all, \tilde{P}_i can be simplified: instead of an j -times encryption of f_i we just use a suitably padded once-encryption of f_i . Thus, we have proven the next result:

Lemma 2. *The amortized communication complexity of the first u updates of the PrivateBDD-based PIR-writing protocol is $O(u\ell \cdot \log n + u^2\kappa \cdot \log^2 n)$, and its computational complexity is dominated by $nm \approx n \cdot \log_2 n$ public-key operations, where the cost of a single public-key operation depends in ℓ , and increases with every update.*

Comparison to Previous Work. This protocol is more communication-efficient than the PIR-writing protocols of [2,3] if $u = o(\sqrt[4]{\ell^{1+\alpha} \cdot n} \cdot \text{polylog}(n))$. Unfortunately, during this protocol the database kept by the server will increase in length, and therefore every new update will be more and more expensive. This can be partially solved by letting the client and the server refresh the database after every (say) $\sqrt[4]{n}$ updates.

However, importantly, this new PrivateBDD-based PIR-writing protocol does *not* use pairings. In particular, it relies on the classical Decisional Composite Residuosity assumption [14] which is by far the weakest security assumption under which a nontrivial PIR-writing protocol is known. Even in the trivial PIR-writing protocol, the database has to be encrypted by using a CPA-secure public-key cryptosystem. It is conceivable that this cryptosystem must be additively homomorphic to allow the client and the server to execute some additional simple cryptocomputing protocols on encrypted database. In such a case, the PrivateBDD-based PIR-writing protocol comes “free” in the sense of security, adding no extra security assumptions.

3.1 Write-Optimized PrivateBDD-Based PIR-Writing Protocol

Clearly, the client’s write requests can be seen as program code that modifies the database. In the previously presented PrivateBDD-based PIR-writing protocol, the server applies the client’s program (BDD) to her database and stores the program outputs as the new database. Unfortunately, the program’s outputs are multiple-encrypted and thus the new database will have longer elements.

Given this interpretation, it is easy to see that one can optimize the presented PIR-writing protocol as follows. The server will not run the client’s program (BDD) at every run of the PIR-writing protocol. Instead, she will append it to the list of previous “programs”. More precisely, the server will just store all client’s messages as follows.

Write-Optimized PrivateBDD-Based PIR-Writing Protocol

Common inputs: Database size n , $m \leftarrow \lceil \log_2 n \rceil$, element length ℓ .

Client’s inputs: Secret key sk , $x = (x_0, \dots, x_{m-1})$, $y \in \{0, 1\}^\ell$.

Server’s inputs: Public key pk , $c = (c_0, \dots, c_{n-1})$, where c_j is an encryption of f_j .

Server’s private output: Updated state $state$. Initially $state$ is empty string.

1. Client sends to the server $C \leftarrow E_{\text{pk}}(x_i)$, for $i \in \{0, m-1\}$, and $D \leftarrow E_{\text{pk}}(y)$.
2. The server does: Set $state \leftarrow state || C$.

In the corresponding CPIR protocol (that reads an element from the database), the client sends his query to the server, who then applies all stored programs to it, and to the database:

CPIR Protocol Corresponding to Write-Optimized PIR-Writing

Common inputs: Database size n , $m \leftarrow \lceil \log_2 n \rceil$, element length ℓ .
Client's inputs: Secret key sk , x .
Server's inputs: Public key pk , $c = (c_0, \dots, c_{n-1})$, where c_j is an encryption of f_j , and $\text{state} = (C_1, D_1) \parallel \dots \parallel (C_u, D_u)$.
Client's private output: Current value of f_x .

1. Client sends to the server the query of CPIR protocol.
2. The server does:
 - (a) For $i \in \{0, \dots, n-1\}$:
 - i. Let $c'_{0,i} \leftarrow c_i$.
 - ii. For $j \in \{1, \dots, u\}$: execute PrivateBDD by using $\tilde{P}_i(x, y, c'_{j-1,i})$, and set $c'_{j,i}$ to be equal to its output.
3. The server executes server's part in the CPIR protocol by using database $c'_u = (c'_{u1}, \dots, c'_{un})$.

Note that this variant achieves very fast writing (and also, good memory efficiency) by offloading computational cost to the reading phase. This solution is good, for example, when the client updates his database often but needs to read its values quite rarely.

4 New FH-Based PIR-Writing Protocol

In this section, we describe another PIR-writing protocol that is based on a fully-homomorphic cryptosystem $\mathsf{P} = (\mathsf{G}, \mathsf{E}, \mathsf{D})$. First, we need the next result.

Lemma 3. *Assume i is known. Let $\text{eq}_i(x) = 1$ if $i = x$, and let $\text{eq}_i(x) = 0$ otherwise. Then a server who knows $\mathsf{E}_{\text{pk}}(x_j)$, for $0 \leq j < m$, can homomorphically evaluate $\mathsf{E}_{\text{pk}}(\text{eq}_i(x))$ by using a circuit of size $m-1$ and of depth $\lceil \log_2 m \rceil$.*

Proof. The circuit has m leaves, labeled by $x_j^{i_j}$ for $0 \leq j < m$, where $x_j^0 = \neg x_j = 1 - x_j$, and $x_j^1 = x_j$. This can be implemented with almost no cost: if $i_j = 1$, then the server inputs $\mathsf{E}_{\text{pk}}(x_j)$ to the gate; if $i_j = 0$, then the server inputs $\mathsf{E}_{\text{pk}}(1 - x_j)$ to the gate. After that the server “AND”-s all m inputs together by using $m-1$ AND-gates arranged in a circuit of depth $\lceil \log_2 m \rceil$, as depicted by Fig. 2. Finally, a 2-fan-in AND-gate can be implemented by setting $\mathsf{E}_{\text{pk}}(a \wedge b) = \mathsf{E}_{\text{pk}}(a) \cdot \mathsf{E}_{\text{pk}}(b)$. \square

The server then cryptocomputes the value of

$$\mathsf{E}_{\text{pk}}(f'_i) \leftarrow (\mathsf{E}_{\text{pk}}(y) - \mathsf{E}_{\text{pk}}(f_i)) \cdot \mathsf{E}_{\text{pk}}(\text{eq}_i(x)) + \mathsf{E}_{\text{pk}}(f_i) , \quad (1)$$

and then replaces $\mathsf{E}_{\text{pk}}(f_i)$ with $\mathsf{E}_{\text{pk}}(f'_i)$.

This idea can be obviously extended to ℓ -bit database elements, although then every bit f_{ij} , $0 \leq j < \ell$, of every database element f_i , $0 \leq i < n$, has to be

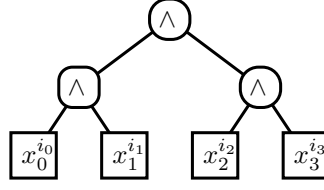


Figure 2. Circuit eq_i for comparing x and i . Here, $n = 16$ and thus $m = 4$. Moreover, $x^i = x$ if $i = 1$, and $x^i = 1 - x$ if $i = 0$

encrypted separately. On the other hand, the circuit eq_i has only to be evaluated once per every $i \in \{0, \dots, n - 1\}$. The full protocol description follows.

FH-Based New PIR-Writing Protocol

Common inputs: Database size n , $m \leftarrow \lceil \log_2 n \rceil$, element length ℓ .

Client's inputs: Secret key sk , $x = (x_0, \dots, x_{m-1})$, $y \in \{0, 1\}^\ell$.

Server's inputs: Public key pk , $c = (c_{00}, \dots, c_{n-1, \ell-1})$ where $c_{ij} = \text{E}_{\text{pk}}(f_{ij})$.

Server's private output: New updated database $c' = (c'_{00}, \dots, c'_{n-1, \ell-1})$.

1. The client sends $(\text{E}_{\text{pk}}(x_0), \dots, \text{E}_{\text{pk}}(x_{m-1}))$ and $(\text{E}_{\text{pk}}(y_0), \dots, \text{E}_{\text{pk}}(y_{\ell-1}))$ to the server.
2. The server does in parallel for $i \in \{0, \dots, n - 1\}$:
 - (a) The server runs encrypted circuit $b_i \leftarrow \text{E}_{\text{pk}}(\text{eq}_i(x))$.
 - (b) For $0 \leq j \leq \ell$, the server computes and stores

$$c'_{ij} \leftarrow (\text{E}_{\text{pk}}(y_j) - c_{ij}) \cdot b_i + c_{ij} .$$

- (c) (If needed,) the server bootstraps all values c'_{ij} to decrease the noise.

Theorem 3. *Assume that the underlying fully-homomorphic cryptosystem is correct and CPA-secure. Then the new FH-based PIR-writing protocol is correct and client-private. Moreover, it has communication complexity $O(\kappa \cdot \log n + \kappa \ell)$, and its computational complexity is dominated by $O(n \cdot \log n + \ell \cdot n)$ applications of Gentry's cryptosystem. (Here we do not count the cost of bootstrapping.)*

Proof (Sketch.). Correctness follows from the correctness of the fully-homomorphic cryptosystem, and from the correctness of the circuit for eq_i . Moreover, since the depth of the circuit used inside the FH-based protocol is only $\approx \log_2 \log_2 n$ (which is never larger than 5 in practice), we can freely assume that there is no need to bootstrap the circuit before the end of the PIR-writing protocol.

The client-privacy of the FH-based PIR-writing protocol follows from the CPA-security of the fully-homomorphic cryptosystem by a standard hybrid argument with $m + \ell$ games. Again, the security is even achieved for computationally bounded malicious server. The efficiency is clear. \square

Comparison to Previous Work. The FH-based protocol has (asymptotically) optimal communication complexity, and differently from the PrivateBDD-based protocol, the database elements do not grow in length after every update. On the other hand, it is based on a fully-homomorphic cryptosystem, and all existing fully-homomorphic cryptosystems rely on relatively new security assumptions. Moreover, the necessary bootstrapping step makes it computationally less efficient.

Further Work. The concept of PIR-writing is somewhat similar to the concept of oblivious RAM, where the distribution of client's read/write queries to the server must be completely independent of the read/write queries the client actually intends to do. Formally speaking, in the oblivious RAM setting, the adversary sits inbetween the client and (honest) server, but has access to all the queries. It is known [12], that one can implement oblivious RAM with a polylogarithmic overhead. We leave it as an interesting open question whether the techniques of [12] can be used to construct (even more) efficient PIR-writing protocols, or our techniques can be used to construct more efficient oblivious RAM protocols.

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

References

1. Boneh, D., Goh, E.J., Nissim, K.: Evaluating 2-DNF Formulas on Ciphertexts. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 325–341. Springer Verlag, Cambridge, MA, USA (Feb 10–12, 2005)
2. Boneh, D., Kushilevitz, E., Ostrovsky, R., Skeith III, W.E.: Public Key Encryption That Allows PIR Queries. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 50–67. Springer-Verlag, Santa Barbara, USA (Aug 19–23, 2007)
3. Chandran, N., Ostrovsky, R., Skeith III, W.E.: Public-Key Encryption with Efficient Amortized Updates. Tech. Rep. 2008/429, International Association for Cryptologic Research (2008), available at <http://eprint.iacr.org/2008/429>
4. Damgård, I., Jurik, M.: A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 119–136. Springer-Verlag, Cheju Island, Korea (Feb 13–15, 2001)
5. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully Homomorphic Encryption over the Integers. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. ?, pp. ?–? Springer-Verlag, Nice, France (May 30–June 3, 2010), to appear

6. Gentry, C.: A Fully Homomorphic Encryption Scheme. Ph.D. thesis, Stanford (Sep 2009)
7. Gentry, C.: Fully Homomorphic Encryption Using Ideal Lattices. In: Mitzenmacher, M. (ed.) STOC 2009. pp. 169–178. ACM Press, Bethesda, MD, USA (May 31 — June 2 2009)
8. Gentry, C., Ramzan, Z.: Single-Database Private Information Retrieval with Constant Communication Rate. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 803–815. Springer-Verlag, Lisboa, Portugal (2005)
9. Ishai, Y., Paskin, A.: Evaluating Branching Programs on Encrypted Data. In: Vadhan, S. (ed.) TCC 2007. LNCS, vol. 4392, pp. 575–594. Springer Verlag, Amsterdam, The Netherlands (Feb 21–24, 2007)
10. Lipmaa, H.: An Oblivious Transfer Protocol with Log-Squared Communication. In: Zhou, J., Lopez, J. (eds.) ISC 2005. LNCS, vol. 3650, pp. 314–328. Springer-Verlag, Singapore (Sep 20–23, 2005)
11. Lipmaa, H.: First CPIR Protocol with Data-Dependent Computation. In: Lee, D., Hong, S. (eds.) ICISC 2009. LNCS, vol. 5984, pp. ?–? Springer-Verlag, Seoul, Korea (Dec 2–4, 2009)
12. Ostrovsky, R.: Efficient Computation on Oblivious RAMs. In: STOC 1990. pp. 514–523. Baltimore, Maryland, USA (14–16 May 1990)
13. Ostrovsky, R., Shoup, V.: Private Information Storage. In: STOC 1997. pp. 294–303 (1997)
14. Paillier, P.: Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer-Verlag, Prague, Czech Republic (May 2–6, 1999)
15. Wegener, I.: Branching Programs and Binary Decision Diagrams: Theory and Applications. Monographs on Discrete Mathematics and Applications, Society for Industrial Mathematics (2000)