First CPIR Protocol with Data-Dependent Computation

Helger Lipmaa[1,2]

[1] Cybernetica AS, Estonia
[2] Tallinn University, Estonia

Abstract We design a new $(n, 1)$-CPIR protocol BddCpir for ℓ-bit strings as a combination of a noncryptographic (BDD-based) data structure and a more basic cryptographic primitive (communication-efficient $(2, 1)$-CPIR). BddCpir is the first CPIR protocol where server's online computation depends substantially on the concrete database. We then show that (a) for reasonably small values of ℓ, BddCpir is guaranteed to have simultaneously log-squared communication and sublinear online computation, and (b) BddCpir can handle huge but sparse matrices, common in data-mining applications, significantly more efficiently compared to all previous protocols. The security of BddCpir can be based on the well-known Decisional Composite Residuosity assumption.
Keywords. Binary decision diagram, computationally-private information retrieval, privacy-preserving data mining, sublinear communication.

1 Introduction

(Single-database) computationally-private information retrieval (CPIR) is one of the most basic cryptographic protocols in the client-server setting. More precisely, in an $(n, 1)$-CPIR protocol, the client retrieves an element chosen by him from server's n-element database of ℓ-bit strings, so that the server obtains no knowledge about which element was transfered. It is always required that the total communication of the CPIR protocol be less than $n\ell$ bits. CPIR protocols constructed in [18,12] are almost optimally communication-efficient. Unfortunately, in all prior nontrivial CPIR protocols, the server's online computational complexity is $\Omega(n)$ public-key operations. Thus, in most of the applications, one is restricted to databases of size say $n = 2^{10}$, which makes computation-efficiency the main bottleneck in deploying CPIR protocols in practice.

In the case of multiple servers, [3] constructed several sublinear-computation information-theoretically private information retrieval protocols. They posed as an open problem to design a sublinear-computation single-server CPIR protocol. This goal has remained so elusive that many researchers have claimed linear computation to be lower-bound for any CPIR, see for example [5, Sect. 1.2], [6, Sect. 2.3] and [12, Sect. 3] for just a few examples. Based on empirical research, Carbunar and Sion [6] argued that in the foreseeable future all linear-communication CPIR protocols will be at least one order of magnitude slower than the trivial CPIR protocol, where the server just transfers the whole database to the client.

Our Contributions. Up to now, one has considered CPIR to be a basic primitive where the server does a fixed amount of work that does not depend on her concrete database. We show that one can efficiently combine noncryptographic data preprocessing with the cryptographic protocol, such that the combination is still secure and at the same time more efficient than the prior work. (In fact it is clear that without preprocessing, the server has to do some online work with every database element.) This in particular shows that $(n, 1)$-CPIR is not a monolithic cryptographic primitive *per se* but can be seen as a combination of a "noncryptographic" data structure (in our case, based on binary decision diagrams) and a more basic cryptographic primitive (in our case, a communication-efficient $(2, 1)$-CPIR). In our opinion, this presents a significant paradigm shift.

Now, let x be the client's index, let $f = (f_0, \ldots, f_{n-1})$ be the server's database. In the new $(n, 1)$-CPIR protocol BddCpir, we write down an optimized BDD for the function f, $f(x) := f_x$, and then use the PrivateBDD protocol [14] to cryptocompute f. We describe an optimized version of it in Sect. 3 in full detail. In particular, [14] assumed that a strong oblivious transfer protocol is used in every node. We show that a $(2, 1)$-CPIR protocol can be used instead. Our variant of the PrivateBDD protocol also has communication that is linear in the length $\mathsf{len}(\mathcal{F})$ of the constructed BDD. More precisely, when using Lipmaa's $(2, 1)$-CPIR protocol from [18], the communication complexity of BddCpir is proportional to $|x| \cdot (|f(x)| + \mathsf{len}(\mathcal{F}))$; see Sect. 3. Server's online computation in BddCpir is dominated by $\mathsf{size}(f)$ public-key operations, where $\mathsf{size}(f)$ is the size of the BDD that corresponds to server's *fixed* input f. This should be contrasted to more general two-party computation protocols where the computation is dominated by the size of the (say) circuit where server's input f is a variable.

After that, we present two different applications. First, for $\ell = 1$, we show that in BddCpir, server's online computational complexity is upperbounded by $(1+o(1))n/\log_2 n$ public-key operations, while the communication complexity is $\Theta(k \cdot \log^2 n)$, where k is the security parameter. The offline computational complexity of this variant of BddCpir is $O(n)$ *non-cryptographic* operations, while setting up the data structure, and $\tilde{O}(t)$, when t elements are updated. Alternatively, this result shows that one can implement secure function evaluation of any $f : \{0,1\}^m \to \{0,1\}$ with communication complexity $O(m^2 \cdot k)$ and server's online computation $O(2^m/m)$. This means that say for databases of size 2^{14}, about 7 times less public-key operations *in the worst case* are needed than in Lipmaa's $(n, 1)$-CPIR from [18]. Importantly, the new protocol has exactly the same communication complexity as Lipmaa's CPIR. In general (and again in the worst case), about 4 to 8 times larger databases can be handled than with Lipmaa's $(n, 1)$-CPIR in the same time, which brings us closer to the practical deployment of $(n, 1)$-CPIR protocols. Moreover, for any ℓ, one can construct a CPIR protocol with communication complexity $\Theta(\ell \cdot \log n + k \cdot \log^2 n)$ and online computation of $\Theta(n\ell/(\log n + \log \ell))$ public-key operations. Thus, if $\ell = o(\log n)$, then BddCpir has still guaranteed sublinear online computation.

However, clearly, the BDD depends on the concrete database. If the databases are well-structured, then one can decrease the computation much more. We show

that if the database is sparse, with $c \ll n$ non-zero elements, the BddCpir protocol has the same communication complexity as before, but its online computational complexity is reduced to $\approx c \cdot \log_2 n$. This version of BddCpir can be used in any of the innumerable privacy-preserving data-mining applications that deal with huge (say, 10 000 times 10 000) but very sparse Boolean matrices. Here, linear-time CPIR protocols are clearly not applicable. However, if the matrix is very sparse, then one can efficiently present the matrix as a BDD, and then apply BddCpir. As an example, the BddCpir protocol can handle $20\,000 \times 20\,000$ permutation matrices about 700 times faster than linear-time CPIR protocols. Moreover, representation as a BDD does not necessarily carry with itself additional cost, since many common data-mining subroutines can be efficiently performed on BDDs [9]. We emphasize that this example is important: the main reason why cryptography-based privacy-preserving data mining has not taken off is the utter inefficiency of existing cryptographic methods in handling huge but structured data. Instead, one uses insecure but severely more efficient methods, see e.g. [1], when processing such data.

In addition, in many existing cryptographic protocols where the server has to cryptocompute some value and then return it to the client, because of the lack of more efficient methods, the server precomputes a database of possible answers and then the client and the server execute an $(n, 1)$-CPIR protocol. In such cases, the database has a clear structure, and thus the BddCpir protocol can be applied.

As a separate contribution, we show how to optimize the PrivateBDD protocol even further. In particular, we present three versions of Lipmaa's $(n, 1)$-CPIR protocol from [18] that have the communication complexity of $2\ell + (2 + o(1)) \log^2 n \cdot k$, $(1 + o(1))\ell + (1 + o(1)) \log^2 n \cdot \log\log n \cdot k$ and $\Theta(\ell \cdot \log n / \log\log n + k \cdot \log^2 n / \log\log n)$, respectively. The balancing techniques are applicable also in the case of the new BddCpir protocol. In particular, the second of those results shows that the new CPIR protocol achieves optimal rate $1 + o(1)$ in the case of a large ℓ, while simultaneously achieving sublinear computation in the case of a large n.

In Sect. 4 we also discuss how to modify BddCpir so that it will also protect server's privacy, that is, to an oblivious transfer (OT) protocol in a virtually costless way.

2 Preliminaries

Client's input is $x \in \{0,1\}^m$, server's input is a function $f : \{0,1\}^m \to \{0,1\}^{\sigma\ell}$ for suitably chosen σ and ℓ. (See the next paragraph for the precise meaning of σ and ℓ, in a concrete application they are chosen so as to minimize the cost of the BddCpir protocol.) We also denote $f(x)$ by f_x, that is, we think of f as of the characteristic function of the vector $f = (f_0, \ldots, f_{2^m-1})$. Also, n denotes the server's database size, and k denotes the security parameter. If A is either a set or a (randomized) algorithm, then $a \leftarrow A$ denotes assignment of a according to the implicit random distribution. All logarithms have base 2.

(Integer-Valued) Binary Decision Diagrams. A *binary decision diagram* (BDD, or a branching program, [24]) is a fanout-2 directed acyclic graph (V, E), where the non-terminal (that is, non-sink) nodes are labeled by variables from some variable set $\{x_0, \ldots, x_{m-1}\}$, the sinks are labeled by ℓ-bit strings and the two outgoing edges of every internal node are respectively labeled by 0 and 1. Usually, it is assumed that a BDD has 1-bit sink labels, then it can be assumed to have two terminal nodes. A BDD with longer sink labels is thus sometimes called *multi-terminal*. A BDD that has σ sources computes some function $f : \{0,1\}^m \to \{0,1\}^{\sigma\ell}$. Every source and every assignment of the variables selects one path from this 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 concatenation of its source values.

In an *ordered binary decision diagram* (OBDD), an order π of the labels is chosen, and for any edge $(u, v) \in E$ it must hold that $\pi(u) < \pi(v)$. A BDD is a *decision tree* if the underlying graph is a tree. A BDD is *layered* if its set of nodes can be divided into disjoint sets V_j such that every edge from a node in set V_j ends in a node in set V_{j+1}. For a BDD P, let $\mathsf{len}(P)$ be its length (that is, the length of its longest path), $\mathsf{size}(P)$ be its size (that is, the number of non-terminal nodes). Let $\mathsf{BDD}(f)/\mathsf{OBDD}(f)$ be the minimal size of any BDD/OBDD computing f. It is known that any Boolean function $f : \{0,1\}^m \to \{0,1\}$ has $\mathsf{BDD}(f) \leq (1+o(1))2^m/m$ [4, Thm. 1] and $\mathsf{OBDD}(f) \leq (2+o(1))2^m/m$ [17,13,4].

Public-Key Cryptosystems. Let $\Pi = (\mathsf{G}, \mathsf{E}, \mathsf{D})$ be a length-flexible additively-homomorphic public-key cryptosystem [7], where G is a randomized key generation algorithm, E is a randomized encryption algorithm and D is a decryption algorithm. in a length-flexible cryptosystem, 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 [7], for every integer $\ell > 0$, $\mathsf{E}_{\mathsf{pk}}(\ell, \cdot) \in \{0,1\}^{\lceil \ell/k \rceil \cdot k + k}$. (In some other length-flexible cryptosystems like [8], the resulting ciphertext is longer.) In practice, $2^\ell < N$ where N is the public key of the DJ01 cryptosystem.

Thus, in the case of the DJ01 cryptosystem, $\mathsf{E}_{\mathsf{pk}}(\ell, M)$ is a valid plaintext of $\mathsf{E}_{\mathsf{pk}}(\lceil \ell/k \rceil \cdot k + k, \cdot)$, and therefore one can multiple-encrypt messages as say in

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

and then recover M by multiple-decrypting,

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

Note that the length of j-times encrypted M is $\lceil \ell/k \rceil \cdot k + jk \leq \ell + (j+1) \cdot k$ bits. 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 also 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 $\mathsf{E}_{\mathsf{pk}}(\ell, M) \in \{0,1\}^{\lceil \ell/k \rceil \cdot k + k}$. As shown in [18] and later in [14], DJ01 has a very simple compress function that just reduces $\mathsf{E}_{\mathsf{pk}}(\ell', M)$ modulo some power of N.

In the CPA (*chosen-plaintext attack*) game, the challenger first generates a random key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{G}(1^k)$, and sends pk to the attacker. The attacker chooses two messages M_0, M_1 and a length parameter ℓ, and sends them to the challenger. The challenger picks a random bit b, and sends a ciphertext $\mathsf{E}_{\mathsf{pk}}(\ell, M_b)$ to the attacker. The attacker outputs a bit b', and wins if $b = b'$. A cryptosystem is *CPA-secure* if the probability that any nonuniform probabilistic polynomial-time attacker wins in the CPA-game is negligibly different from $1/2$.

Clearly, because of the existence of the compress function, a CPA-secure length-flexible cryptosystem remains CPA-secure even if the adversary sends many message pairs (M_{j0}, M_{j1}) and length parameters ℓ_j, and has to guess b after seeing encryptions of all M_{jb} under the corresponding length parameters ℓ_j. This so-called LFCPA-security [18] of the cryptosystem is crucial for the security of the efficient PrivateBDD protocol as defined in the next section. The DJ01 cryptosystem [7] is CPA-secure under the Decisional Composite Residuosity Assumption [22].

CPIR. In a 1-out-of-n computationally-private information retrieval protocol, $(n,1)$-CPIR, for ℓ-bit strings, the client has an index $x \in \{0, \ldots, n-1\}$ and the server has a database $f = (f_0, \ldots, f_{n-1})$ with $f_i \in \{0,1\}^{\ell}$. The client obtains f_x. The new $(n,1)$-CPIR protocol BddCpir, proposed in this paper, is based on an $(2,1)$-CPIR protocol that satisfies some very specific requirements. Namely, we say that an $(n,1)$-CPIR protocol $\Gamma = (\mathsf{Q}, \mathsf{R}, \mathsf{A}, \mathsf{C})$ is *BDD-friendly* if it satisfies the next four assumptions:

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

That is, $\Gamma = (\mathsf{Q}, \mathsf{R}, \mathsf{A}, \mathsf{C})$ is a quadruple of probabilistic polynomial-time algorithms, with $\mathsf{A}(\ell, x, \mathsf{R}(\ell, f, \mathsf{Q}(\ell, x))) = f_x$, and $\mathsf{C}(\ell', \ell, \mathsf{Q}(\ell', x)) = \mathsf{Q}(\ell, x)$ for any $\ell' \geq \ell$, x and f. For related work on computation-efficient CPIR protocols, see for example [10,2].

Let $\Pi = (\mathsf{G}, \mathsf{E}, \mathsf{D})$ be a length-flexible additively homomorphic public-key cryptosystem. Client's private input is $x \in \{0,1\}$, server's private input is $f = (f_0, f_1)$ for $f_0, f_1 \in \{0,1\}^{\ell}$. In [18], Lipmaa proposed a $(2,1)$-CPIR protocol that consists of the next three steps:

1. The client sets $(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{G}(1^k)$, $c \leftarrow \mathsf{E}_{\mathsf{pk}}(\ell, x)$, and sends $\mathsf{Q}(\ell, x) \leftarrow (\mathsf{pk}, c)$ to the server.
2. The server replies with $\mathsf{R} = \mathsf{R}(\ell, f, (\mathsf{pk}, c)) \leftarrow \mathsf{E}_{\mathsf{pk}}(\ell, f_0) \cdot c^{f_1 - f_0}$.
3. The client outputs $\mathsf{A}(\ell, x, \mathsf{R}) := \mathsf{D}_{\mathsf{sk}}(\ell, \mathsf{R})$.

If $x \in \{0,1\}$, then clearly

$$\begin{aligned}\mathsf{R}(\ell, f, (\mathsf{pk}, \mathsf{E}_{\mathsf{pk}}(\ell, x))) =& \mathsf{E}_{\mathsf{pk}}(\ell, f_0) \cdot c^{f_1 - f_0} = \mathsf{E}_{\mathsf{pk}}(\ell, f_0) \cdot \mathsf{E}_{\mathsf{pk}}(\ell, x)^{f_1 - f_0} \\ =& \mathsf{E}_{\mathsf{pk}}(\ell, f_0 + (f_1 - f_0) \cdot x) = \mathsf{E}_{\mathsf{pk}}(\ell, f_x) \ .\end{aligned}$$

If Π has a compress function, then Lipmaa's $(2,1)$-CPIR protocol has also a compress function C that just compresses both involved ciphertexts. Importantly, $|\mathsf{Q}(\ell,\cdot)|, |\mathsf{R}(\ell,\cdot,\cdot)| \le \ell + 2k$ and thus, this $(2,1)$-CPIR protocol is BDD-friendly.

Semisimulatable Privacy. Let $\Gamma = (\mathsf{Q}, \mathsf{R}, \mathsf{A}, \mathsf{C})$ be a 2-message $(n,1)$-CPIR protocol. As many previous papers [21,18,14], we only require (semisimulatable) privacy in the malicious model. More precisely, client's privacy is guaranteed in the sense of indistinguishability (CPA-security), while server's privacy is guaranteed (if at all) in the sense of simulatability. This assumption makes it possible to design 2-message $(n,1)$-CPIR protocols that are both communication and computation-efficient. We now give an informal definition of privacy.

For the *CPA-security* (that is, the privacy) *of the client*, no malicious nonuniform probabilistic polynomial-time server should be able to distinguish, with non-negligible probability, between the distributions $\mathsf{Q}(\ell, x_0)$ and $\mathsf{Q}(\ell, x_1)$ that correspond to any two of client's inputs x_0 and x_1 that are chosen by herself. For *server-privacy*, we require the existence of an unbounded simulator that, given client's message Q^* and client's legitimate output corresponding to this message, generates server's message that is statistically indistinguishable from server's message R in the real protocol; here Q^* does not have to be correctly computed. A protocol is *private* if it is both client-private and server-private.

Any $(n,1)$-CPIR protocol Γ must be client-private, that is, CPA-secure. Lipmaa's $(2,1)$-CPIR protocol [18], when based on the DJ01 cryptosystem [7], is CPA-secure under the DCR Assumption [22]. Because of the existence of the compression function, if Γ is CPA-secure then it is also difficult to distinguish between any two polynomially large sets $\{\mathsf{Q}(\ell_i, x_{i0})\}$ and $\{\mathsf{Q}(\ell_i, x_{i1}))\}$, even if the same public key pk is used in all of them. A private $(n,1)$-CPIR protocol is also known as an $(n,1)$-*oblivious transfer protocol*.

3 The PrivateBDD Protocol

Next, we describe the PrivateBDD cryptocomputing protocol from [14]. It generalizes the cryptocomputing process, done in several previous $(n,1)$-CPIR protocols [15,23,18]. Our exposition is simpler than the more general exposition of [14]. The concrete protocol has also some small differences compared to the

protocol of [14]. More precisely, while the description given by us can be inferred from the description in [14], we have opted to describe explicitly the most efficient known implementation of the PrivateBDD. Moreover, Ishai and Paskin [14] used a strong oblivious transfer protocol at every node of the underlying BDD, while we just use an efficient $(2,1)$-CPIR protocol. There are also other minor differences.

In the PrivateBDD protocol for some set $\mathcal{F}$ of functions, the client has private input $x \in \{0,1\}^m$, the server has private input $f : \{0,1\}^m \to \{0,1\}^{\sigma\ell}$ with $f \in \mathcal{F}$, and the client will receive private output $f(x)$. Here, $\mathcal{F} = \{f : \{0,1\}^m \to \{0,1\}^{\sigma\ell}\}$ is a set of functions, where every $f \in \mathcal{F}$ can be computed by some polynomial-size BDD P_f that has σ sources and ℓ-bit sink labels. Define $\mathsf{len}(\mathcal{F}) := \max_{f\in\mathcal{F}} \mathsf{len}(P_f)$. Let $\Gamma' = (\mathsf{Q}', \mathsf{R}', \mathsf{A}', \mathsf{C}')$ be a BDD-friendly $(2,1)$-CPIR protocol. Since we are going to recursively apply Γ' on databases that consist of the R' values of some other runs of Γ', we need to define the next few values. Namely, let

$$\begin{aligned} |\mathsf{Q}^{(1)}(\ell)| &:= |\mathsf{Q}'(\ell, x)| \ , \\ |\mathsf{R}^{(j)}(\ell)| &:= |\mathsf{R}'(|\mathsf{Q}^{(j)}(\ell)|, f, \mathsf{Q}')| \ , \\ |\mathsf{Q}^{(j+1)}(\ell)| &:= |\mathsf{Q}'(|\mathsf{R}^{(j)}(\ell)|, x)| \ . \end{aligned}$$

We will assume that those values are well-defined, that is, that they do not depend on the concrete values of x and f. Because Γ' has to be private, this assumption is reasonable. If Γ' is BDD-friendly, then $|\mathsf{Q}^{(j)}(\ell)| = |\mathsf{R}^{(j)}(\ell)| \leq \ell + j \cdot \Theta(k)$.

Now, BDDs are usually evaluated in a top-down manner by following the σ paths that are consistent with the assignment of the input variables x_j. It is unlikely that one can evaluate BDDs like this in a private manner. Instead, following [14], we use a bottom-up way of evaluating a BDD. In the non-private version of this process, the sinks' output values are equal to their labels. At every non-terminal node v that is labeled by some x_j and for which the output values R_{v_0} and R_{v_1} of both children are known, one sets the output value R_v of v to be equal to $\mathsf{R}_{v_{x_j}}$. The value of the BDD is equal to the concatenation of the output values of the sources.

In the private version, the server also executes the BDD P_f bottom-up, that is, starting from the sinks. The output values R_v of the sinks are equal to their ℓ-bit labels. Initially, R_v is undefined for all other nodes. At every node v of the BDD with label x_j and children v_0/v_1 such that the output values $\mathsf{R}_{v_0}/\mathsf{R}_{v_1}$ of v_0/v_1 are known but the output value R_v of v is not yet defined, the server uses Γ to obliviously propagate the value $\mathsf{R}_{v_{x_j}}$ upwards as R_v. The server does this for all nodes in some ordering, and then sends the output values of the σ sources to the client. (Ishai and Paskin [14] only considered the depth-first ordering, while sometimes some other ordering may be more efficient.) For every source, the client applies the decoding procedure A' repeatedly to obtain the label of the sink that is uniquely determined by this source and by client's input x. Complete description of the PrivateBDD protocol for $\mathcal{F}$ is given by Protocol 1.

1. **Common inputs:** $m, \sigma, \ell, \mathcal{F}, \mathsf{len}(\mathcal{F})$.
2. **Private inputs:** the server has a function $f : \{0,1\}^m \to \{0,1\}^{\sigma\ell}$ from $\mathcal{F}$, and the client has bitstring $x \in \{0,1\}^m$.
3. **Offline phase:** server computes an efficient BDD P_f for f that has σ sources and ℓ-bit sink labels, and where $\mathsf{len}(P_f) \le \mathsf{len}(\mathcal{F})$. Let $\ell_{\max} := |\mathsf{Q}^{(\mathsf{len}(\mathcal{F})-1)}(\ell)|$.
4. **Online phase:**
   (a) **Client does:** For $j \in \{0, \ldots, m-1\}$, set $\mathsf{Q}_j \leftarrow \mathsf{Q}'(\ell_{\max}, x_j)$. Send $\mathsf{Q}(\ell, x) \leftarrow (\mathsf{Q}_0, \ldots, \mathsf{Q}_{m-1})$ to the server.
   (b) **Server does:**
      i. For all sinks v of P_f, set R_v to be their label. For non-terminal nodes v, set $\mathsf{R}_v \leftarrow \bot$.
      ii. Do by following some ordering of the nodes:
         A. Let v be some node with $\mathsf{R}_v = \bot$, with children v_0 and v_1 that have $\mathsf{R}_{v_0}, \mathsf{R}_{v_1} \neq \bot$; if no such node exists then exit the loop.
         B. Assume that v is labeled by x_i and edges from v to v_0/v_1 are labeled by 0/1.
         C. Compute and store $\mathsf{R}_v \leftarrow \mathsf{R}(\ell^*, (\mathsf{R}_{v_0}, \mathsf{R}_{v_1}), \mathsf{C}(\ell_{\max}, \ell^*, \mathsf{Q}_i))$, where $\ell^* \leftarrow \max(|\mathsf{R}_{v_0}|, |\mathsf{R}_{v_1}|)$. // If BDD is layered then $|\mathsf{R}_{v_0}| = |\mathsf{R}_{v_1}|$.
      iii. For all σ sources v, send R_v to the client.
   (c) **Client does:** For any source v, compute private output from R_v by applying A' recursively up to $\mathsf{len}(\mathcal{F})$ times.

Protocol 1: The PrivateBDD protocol

Theorem 1. *Let $\Gamma' = (\mathsf{Q}', \mathsf{R}', \mathsf{A}', \mathsf{C}')$ be a CPA-secure BDD-friendly $(2,1)$-CPIR protocol. Let $\mathcal{F}$ be a set of functions from $\{0,1\}^m$ to $\{0,1\}^{\sigma\ell}$ where every $f \in \mathcal{F}$ can be computed by a polynomial-size BDD P_f. Then $\mathcal{F}$ has a CPA-secure cryptocomputing protocol with the communication complexity $m \cdot |\mathsf{Q}^{(\mathsf{len}(\mathcal{F}))}(\ell)| + \sigma \cdot |\mathsf{R}^{(\mathsf{len}(\mathcal{F}))}(\ell)| = (m+\sigma)(\ell + \mathsf{len}(\mathcal{F}) \cdot k)$. Server's online computation is dominated by $\mathsf{size}(P_f)$ public-key operations. Additionally, if P_f is layered, then the PrivateBDD protocol is server-private in the semihonest model.*

Proof. CPA-security follows by a standard hybrid argument from the LFCPA-security of Γ', and thus from the CPA-security of Γ' and from the existence of C'. If P_f is layered, then the client is completely oblivious to the shape of the BDD, except the length of it: he just forms queries corresponding to his input bits by using his knowledge of the length of the BDD (and on the output length ℓ), and then receives multiple-"encryptions" of the outputs. The communication complexity part is straightforward. Server has to compute R at every node of P_f. □

Alternatively, this theorem shows that if any $f : \{0,1\} \to \{0,1\}^{\sigma\ell}$ has an "efficient" cryptocomputing protocol, then any $F : \{0,1\}^m \to \{0,1\}^{\sigma\ell}$ has an "efficient" cryptocomputing protocol.

If the compress function C does not exist, then the client has to submit up to $\mathsf{len}(P)$ different queries $\mathsf{Q}(\ell', x_j)$ for every x_j and every $\ell' = |\mathsf{Q}^{(i)}(\ell)|$ for

$i \leq \mathsf{len}(P) - 1$. This can increase the communication by a factor of $\mathsf{len}(P)$. The existence of C makes it possible to compute $\mathsf{Q}(|\mathsf{Q}^{(i)}(\ell)|, x_j)$ from $\mathsf{Q}(\ell_{\max}, x_j)$.

We assume throughout this paper that we are working with Lipmaa's $(2,1)$-CPIR from [18], which is currently the only known $(2,1)$-CPIR protocol that allows the PrivateBDD protocol to achieve the communication complexity that is polynomial in $\mathsf{len}(\mathcal{F})$ (thus the name "BDD-friendly"). A precise result follows:

Corollary 1. *Assume that the DCR Assumption [22] is true. Let $\mathcal{F}$ be a set of functions $f : \{0,1\}^m \to \{0,1\}^{\sigma\ell}$, and for any $f \in \mathcal{F}$ let P_f be some σ-source polynomial-size BDD with ℓ-bit sink labels that computes f. Then $\mathcal{F}$ has a CPA-secure cryptocomputing protocol with the communication upperbounded by $k + (m+\sigma) \cdot (\ell + (\mathsf{len}(\mathcal{F}) + 2) \cdot k)$, and server's online computation dominated by $\mathsf{size}(\mathcal{F})$ public-key operations.*

Proof. Let $\Pi = (\mathsf{G}, \mathsf{E}, \mathsf{D})$ be the DJ01 length-flexible cryptosystem [7]. This version of the PrivateBDD protocol generates one single $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{G}(1^k)$ and uses the same pk to construct all m queries Q_j. Because Lipmaa's $(2,1)$-CPIR is BDD-friendly and CPA-secure, the CPA-security of PrivateBDD follows from a standard hybrid argument. Computation-efficiency is straightforward. To calculate the communication efficiency, note that $\mathsf{Q}_j = \mathsf{Q}'(\ell_{\max}, x_j) = \mathsf{E}_{\mathsf{pk}}(\ell + \mathsf{len}(\mathcal{F}) \cdot k, x_j)$. Thus,

$$|\mathsf{Q}_j| = |\mathsf{E}_{\mathsf{pk}}(\ell + \mathsf{len}(\mathcal{F}) \cdot k, x_j)| = (\lceil \ell/k \rceil + \mathsf{len}(\mathcal{F}) + 1) \cdot k \leq \ell + (\mathsf{len}(\mathcal{F}) + 2) \cdot k \ .$$

Therefore, the client sends a public key (of length say k) and at most $m \cdot (\ell + (\mathsf{len}(\mathcal{F}) + 2) \cdot k)$ additional bits. The output of the BDD is equal to σ $(\leq \mathsf{len}(\mathcal{F}))$-times encryptions of sink values, where the sinks are selected by the encrypted client inputs x_j. Server's communication consists of σ $(\leq \mathsf{len}(\mathcal{F}))$-times encrypted messages of length $\leq \ell + (\mathsf{len}(\mathcal{F}) + 2) \cdot k$. □

All CPIR protocols that follow the Kushilevitz-Ostrovsky recursion technique [15,23,18] can be seen as using PrivateBDD to cryptocompute an ordered n'-ary decision tree, with $\sigma = 1$, $m = \lceil \log_{n'} n \rceil$ and varying values of n'. However, in the case of [15,23], the underlying $(n', 1)$-CPIR protocol is not very efficient and thus the communication-complexity of the resulting $(n,1)$-CPIR protocols of [15,23] is not polylogarithmic. On the other hand, Lipmaa's $(n,1)$-CPIR protocol from [18] uses his $(2,1)$-CPIR protocol in combination with an ordered binary decision tree, to achieve the communication complexity $\Theta(m \cdot (\ell + \mathsf{len}(P_f) \cdot k)) = \Theta(\ell \cdot \log n + k \cdot \log^2 n)$, agreeing with Cor. 1. Note that such CPIR protocols do not explicitly need the C function, because they cryptocompute an ordered binary decision tree where every x_i is only tested on the ith level of the tree. More generally, the C function is not necessary if the underlying BDD is ordered.

4 New Computation-Efficient $(n,1)$-CPIR Protocol

Assume that $\sigma = 1$ and that the database size is $n = 2^m$, that is, that the server's database consists of $n = 2^m$ bits. (If n is not a power of 2 then one can

round up the database size by using additional dummy elements.) In this case, we can restate the goals of an $(n, 1)$-CPIR protocol as follows.

Assume that the client has an input $x \in \{0,1\}^m$, and that the server's input is a Boolean function $f : \{0,1\}^m \to \{0,1\}^\ell$, such that $f(x) = f_x$. The client needs to retrieve $f(x)$. Thus in this case, $\mathcal{F}$ is the set of all functions, $\mathcal{F} = \{f : \{0,1\}^m \to \{0,1\}^\ell\}$. In the new $(n, 1)$-CPIR protocol, the client and the server run PrivateBDD for this $\mathcal{F}$. That is:

- In the offline phase of BddCpir, the server computes and stores an efficient BDD P_f for the concrete f.
- In the online phase of BddCpir, the client and the server follow PrivateBDD as specified in Protocol 1. Here, the server uses P_f.

In BddCpir, server's online computational complexity is proportional to $\mathsf{size}(P_f)$ while the communication complexity is proportional to $\mathsf{len}(\mathcal{F})$. We emphasize once more that P_f is computed after server's input f has been fixed.

The size (and to a lesser extent, also the length) of P_f will depend heavily on f (and $\mathcal{F}$), and not much can be said about it unless we know the concrete database or at least some of its properties. In what follows, we will consider two different database classes. First, we look at the case of arbitrary databases. We show that for any possible database f, as long as $\ell = o(\log n)$, the new $(n, 1)$-CPIR protocol has server's computation upperbounded by $o(n)$ public-key operations. Second, we look at the case where it is known that f is a very sparse database (like in many privacy-preserving data mining applications). We show than in the case, BddCpir is computationally significantly more efficient than any other existing CPIR protocol.

4.1 Class 1: Arbitrary Databases

In [4], it was shown that any Boolean function f can be computed by a BDD P_f of size $(1+o(1))2^m/m$ and length $(1+o(1))m$. However, this construction is reasonably efficient only when $m \geq 25$. Instead, we will describe an OBDD $\mathsf{WP}(f)$ from [17,13,4] that meets the upperbound $\mathsf{OBDD}(P_f) \leq (2 + o(1))2^m/m$. This OBDD also has the benefit of having optimal length m. Based on this result, even if f is an arbitrary Boolean database, the BddCpir protocol has communication complexity $\Theta(k \cdot \log^2 n)$ and server's online computational complexity $\Theta(n/\log n)$.

Let $f : \{0,1\}^m \to \{0,1\}$ be a Boolean function. (See Fig. 1 for the concrete case $m = 6$ of the next general construction.) Prot. 2 describes the corresponding OBDD $\mathsf{WP}(f)$, as found in say [24]. Briefly, the idea of $\mathsf{WP}(f)$ is to first branch according to first d variables. After that, the number of possible subfunctions on the last $m - d$ variables will be sufficiently small, so that one can branch according to corresponding subfunctions.

Clearly, this OBDD computes f and has length m. The size of $\mathsf{WP}(f)$ depends on d. There are two different recommendations for d. In [4], it was recommend

- The BDD starts out as a depth-d, where d is fixed later, ordered binary decision tree where one branches on variables $x_0, \ldots, x_{d-1}$. This part of the BDD has $2^d - 1$ nodes.
- The BDD has $2^{2^{m-d}}$ more nodes that correspond to all subfunctions g of f on its last $m-d$ variables. These extra nodes are layered in $m-d$ more levels. The node for a subfunction that first essentially depends on the jth variable out of these $m-d$ variables (but not on earlier ones) is on level $d+j$; nodes that correspond to constant subfunctions are on level m. The extra nodes are labeled by corresponding subfunctions g. Note that the $m-d$ lowest levels have 2, $2^2 - 2^1 = 2$, $2^4 - 2^2 = 12$, $\ldots$, $2^{2^{m-d}} - 2^{2^{m-d-1}}$ nodes respectively.
- Let v_g be an extra (non-terminal) node. Assume that $g(y_1, \ldots, y_{m-d})$ first essentially depends on y_j. For $i \in \{0, 1\}$, let $g_{|y_j=i}$ be the function that we get from g when we set $y_j \leftarrow i$. Add an i-edge from v_g to $v_{g_{|y_j=i}}$.
- The above part of the construction only depends on the value of $n = 2^m$ and not on the concrete database. The next part depends on the database: The 2^{d-1} nodes on level d are labeled by subsequent $2^{m-d+1} = 2 \cdot 2^{m-d}$ values of the 2^m-bit database f. For a fixed level d node v', consider the first 2^{m-d} bits of this label to be the truth table of some subfunction g_0, and the last 2^{m-d} bits to be a truth table of some subfunction g_1. Add a 0-edge from v' to v_{g_0} and a 1-edge from v' to v_{g_1}.

Protocol 2: The description of $\mathsf{WP}(f)$

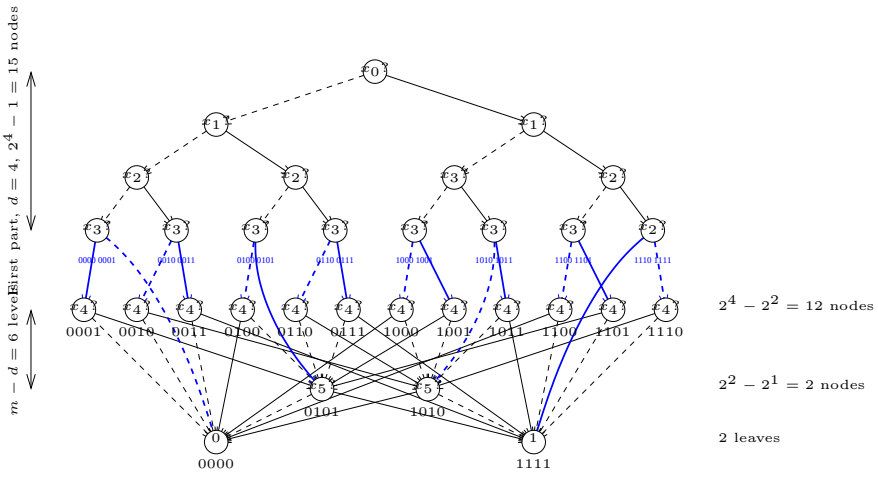

Figure1. Pictorial representation of OBDD corresponding to the presented upper-bound $(2 + o(1))2^m/m$ for $n = 2^m = 64$ and $d = 4$ computed according to Eq. (2). Only blue values and edges depend on the concrete database, which is equal to a sequence of binary presentations of all 4-bit integers. Everything else depends just on the value of m. For the sake of simplicity, we use the truth tables of corresponding subfunctions to label the extra nodes. The concrete database is $f = (0, 0, 0, 0; 0, 0, 0, 1; 0, 0, 1, 0; 0, 0, 1, 1; 0, 1, 0, 0, \ldots)$.

to fix

$$d := m - \lfloor \log_2(m - 2\log_2 m) \rfloor \ . \tag{1}$$

For such d, as it is shown in [4], $\mathsf{WP}(f)$ has size upperbounded by $(2+o(1))2^m/m$. However, for this choice of d to work, one has to assume that $m \geq 7$. Therefore, if m is small, we follow the recommendation of [24] to take

$$d := m - \lfloor \log_2(m + 1 - \log_2 m) \rfloor \ . \tag{2}$$

It is known [24] that with this choice of d, the size of $\mathsf{WP}(f)$ is $(3 + o(1))2^m/m$.

Example 1. If $m = 6$, then $d = 4$ according to Eq. (2). The complete ordered decision tree (which corresponds to the use of Lipmaa's CPIR protocol from [18]) has $2^m - 1 = 63$ non-terminal nodes. The OBDD $\mathsf{WP}(f)$ has $2^d - 1 + 2^{2^{m-d}} - 2 = 15 + 16 - 2 = 29$ non-terminal nodes. Thus even in this pet case, the BddCpir protocol requires $63/29 \approx 2$ times less public-key operations than Lipmaa's CPIR protocol. See Tbl. 1 for $2^m - 1$ and the size of $\mathsf{WP}(f)$ for other values of m, where an optimal d has been numerically optimized. (Note that this usually agrees with d computed according to Eq. (2).) There we see that for $m = 20$, the BddCpir protocol requires 8 times less public-key operations than Lipmaa's CPIR protocol.

We emphasize that it is natural to compare the efficiency of the new BddCpir protocol and Lipmaa's CPIR from [18] in the number of public-key operations since in both cases, one uses the same underlying public-key primitive on the plaintexts of the same length. Thus, one can expect that the actual running time of the BddCpir protocol (measured in seconds) is also about 4 to 8 smaller than the actual running time of Lipmaa's CPIR protocol, while having exactly the same communication complexity.

Now, let us proceed to compute the efficiency of this variation of the BddCpir protocol. *Offline* computation of the BddCpir protocol (the construction of the OBDD that corresponds to the upperbound) takes $O(2^m)$ *non-cryptographic* operations. This value is not so important because the offline computation has to be only done once per database, and not once per query. As evident from the construction of $\mathsf{WP}(f)$, only the location of $2^d \approx 2^m/(m + 1 - \log_2 m) = (1 + o(1)) \cdot 2^m/m$ edges depends on the database. Thus even when the database is completely changed, one has to change $O(2^d) = O(2^m/m)$ edges. This takes $O(2^m/m)$ time in the RAM model, and can be compared to the 2^m work that is necessary to update the database itself. In the case t elements of the database are updated, exactly the location of t edges is changed.

Online evaluation of the BDD $\mathsf{WP}(f)$ on concrete input x takes $(3 + o(1))2^m/m$ public-key operations. As depicted by Tbl. 1, this is smaller than the trivial $n = 2^m$ for any $m \geq 3$. To the best of our knowledge, this is the first $(n, 1)$-CPIR with this property. Note that the upperbound $\Theta(2^m/m)$ is also tight because there exist functions f with $\mathsf{BDD}(f) = (1 - o(1))2^m/m$ [4].

m	2^m-1	$\mathsf{WP}(f)$	Opt. d	Imprv.	m	2^m-1	$\mathsf{WP}(f)$	Opt. d	Imprv.
1	1	1	1	1.0	13	8 191	1 277	10	6.41425
2	3	3	1	1.0	14	16 383	2 301	11	7.11995
3	7	5	2	1.4	15	32 767	4 349	12	7.53438
4	15	9	3	1.66667	16	65 535	8 445	13	7.76021
5	31	17	4	1.82353	17	131 071	16 637	14	7.87828
6	63	29	4	2.17241	18	262 143	33 021	15	7.93868
7	127	45	5	2.82222	19	524 287	65 789	16	7.96922
8	255	77	6	3.31169	20	1 048 575	131 069	16	8.00018
9	511	141	7	3.62411	21	2 097 151	196 605	17	10.6668
10	1 023	269	8	3.80297	22	4 194 303	327 677	18	12.8001
11	2 047	509	8	4.02161	23	8 388 607	589 821	19	14.2223
12	4 095	765	9	5.35294	24	16 777 215	1 114 109	20	15.0589

Table1. The comparison of the size of the binary decision tree and $\mathsf{WP}(f)$

Theorem 2. *Assume that the DCR Assumption holds. Then there exists a CPA-secure $(n,1)$-CPIR protocol for 1-bit strings with the communication complexity $\Theta(\log^2 n)\cdot k$ and server's online computation of $O(n/\log n)$ public-key operations.*

Proof. Follows from Cor. 1 and the upperbound of [17,13,4], by letting $\mathcal{F}$ to be the set of all Boolean functions $f:\{0,1\}^{\lceil \log_2 n\rceil}\to\{0,1\}$, and using the OBDD $\mathsf{WP}(f)$. □

Now, let $f:\{0,1\}^m\to\{0,1\}^\ell$ for some $\ell\geq 1$. By the already mentioned upperbound of [17,13,4], clearly $\mathsf{BDD}(f)\leq \ell\cdot(2+o(1))2^m/m$ by just evaluating ℓ BDDs in parallel. Thus, there exists a sublinear-computation CPIR protocol for say any $\ell\leq m/3$. However, we can prove the next more precise result.

Theorem 3. *(1) Let $f:\{0,1\}^m\to\{0,1\}^\ell$ for some $\ell\geq 1$. For $\ell\geq 1$, $\mathsf{OBDD}(f)\leq (2+o(1))\cdot 2^m\cdot\ell/(m+\log_2\ell)$. (2) Assume that the DCR Assumption holds. There exists a CPA-secure $(n,1)$-CPIR protocol for ℓ-bit strings with the communication complexity $\Theta(\ell\cdot\log n+k\cdot\log^2 n)$ and online computation of $O(\ell\cdot n/(\log n+\log\ell))$ public-key operations.*

Proof (Of Thm. 3). We follow the same ideas as in constructing $\mathsf{WP}(f)$. The new OBDD starts with a complete binary tree of depth d and then has $2^{\ell\cdot 2^{m-d}}$ extra nodes that correspond to all possible subfunctions $f':\{0,1\}^{m-d}\to\{0,1\}^\ell$, with 2^ℓ of those extra nodes being the sinks. The edges are added in the natural way. Thus, this BDD has $2^d-1+2^{\ell 2^{m-d}}-2^\ell$ non-terminal nodes. This value is (almost) minimized when $d=\ell\cdot 2^{m-d}$, that is, when $d=W(2^m\ell\ln 2)/\ln 2$. Here $W(x)$ is the Lambert's W-function, that is the inverse function of $f(w)=w\cdot\exp(w)$. Using this value of d, we get that the constructed OBDD has then

$$2\cdot\exp(W(2^m\ell\cdot\ln 2))-2^\ell-1$$

non-terminal nodes. Next, we use the first two elements of the series expansion of $W(z) = \ln z - \ln\ln z + \ldots$, to find that the constructed OBDD has approximately

$$2 \cdot \exp(\ln(2^m \ell \ln 2) - \ln\ln(2^m \ell \ln 2)) - 2^\ell - 1 = \frac{2^{m+1}\ell}{m + \log_2 \ell + \log_2 \ln 2} - 2^\ell - 1$$
$$\leq 2 \cdot \frac{2^m \ell}{m + \log_2 \ell}$$

non-terminal nodes. Because d has to be integral, the computations are not precise, and there will be a small additional multiplicative constant $1 + o(1)$ that this expression will be multiplied with. Note that the size of this OBDD is smaller than the trivial $2^m - 1$ if say $\ell \leq (m + \log_2 \ell)/3$ or say $\ell \leq (m + \log m)/3$. □

4.2 Case 2: CPIR for Sparse Matrices

In almost all real life data, there is a lot of redundancy. Otherwise, most of the existing data-mining and machine learning algorithms would not be useful in practice. As a concrete application area of the BddCpir protocol, consider privacy-preserving data mining scenarios that often deal with huge (say, 20 000 times 20 000) but very sparse Boolean matrices. For example, in such applications every row of this matrix could be a transaction (say in a supermarket) and every column would correspond to some item sold in this supermarket. An element m_{ij} of this matrix would be 1 exactly when during the ith transaction the jth item was actually bought.

One of the most basic operations in such applications is private retrieval of a single matrix element. Clearly, linear-time CPIR protocols are not applicable in this case due to the raw size of the matrices. However, if the matrix is very sparse, then one can efficiently present the matrix as a BDD (as was recommended say in [9]), and then apply the BddCpir protocol. As noted in [9], BDD is a good data structure for representing sparse matrices. In particular, if the matrix is very sparse, then the next straightforward OBDD representation is already good enough. Namely, assume that the Boolean matrix has dimension $n_1 \times n_2$ and contains $c \ll n_1 n_2$ ones. We can then represent the matrix as a join of c paths of length $\lceil \log_2 n_1 + \log_2 n_2 \rceil$, where every sink (and thus every path) corresponds to exactly one 1 entry in the matrix. Thus, the size of this OBDD representation is upperbounded by $c \cdot \lceil \log_2 n_1 + \log_2 n_2) \rceil$ [9] while its length is upperbounded by $\lceil \log_2 n_1 + \log_2 n_2 \rceil$. This is an upperbound, since all paths share at least one (and usually more) nodes. Thus, in the case of sparse but huge matrices, we can use this trivial representation and then just apply BddCpir to this. Note that, as shown in [9], many matrix algorithms can be performed efficiently on the OBDD representation of sparse matrices, which makes the OBDD representation of sparse matrices reasonable in many data-mining applications and thus one could apply more complex privacy-preserving operations on the top of CPIR.

As a concrete example, assume that we have a $20\,000 \times 20\,000$ matrix. If this matrix has exactly one 1 in every row (like the permutation matrices), then

BddCpir has server's online computation dominated by $\leq 20\,000 \cdot 2\log_2(20\,000) \approx 2^{19.1}$ public-key operations, while all previous CPIR protocols need $20\,000^2 \approx 2^{28.6}$ public-key operations. If the matrix has say 20 ones in every row in average (this is typical in shopping-basket applications), then BddCpir is still about 35 times faster than CPIR protocols with linear computation.

More generally, we have the next result.

Theorem 4. *Assume that the DCR Assumption holds. Assume that $f = (f_0, \ldots, f_{n-1})$ is a sparse (not necessary Boolean) database that has $c \ll n$ non-zero entries. Then there exists a CPA-secure $(n, 1)$-CPIR protocol for ℓ-bit strings with the communication complexity $\Theta(\ell \cdot \log n + k \cdot \log^2 n)$ and server's online computation of $\approx c \cdot \log_2 n$ public-key operations.*

Proof (Sketch.). Similarly to the sparse matrix case, one can construct a trivial BDD with c sinks and paths that has length $\approx \log_2 n$ and size $\approx c \cdot \log_2 n$. According to Cor. 1, in this case BddCpir has communication complexity $\Theta(\ell \cdot \log n + k \cdot \log^2 n)$, and server's online computation is dominated by $\approx c \cdot \log_2 n$ public-key operations. □

In particular, if $c = \Theta(\sqrt{n})$ as in the sparse matrix case, then server's online computation is dominated by $O(\sqrt{n} \cdot \log_2 n)$ public-key operations. Moreover, if $c = n$, then we actually have a complete binary decision tree, which corresponds to the CPIR of [18], and therefore this solution is never less efficient than [18].

5 Discussions

Server-Privacy. Recall that an $(n, 1)$-CPIR protocol that also achieves server-privacy is usually called an $(n, 1)$-OT protocol. As said earlier, as the minimum, the underlying BDD has to be layered or otherwise the protocol will not preserve server's privacy. Most of the BDDs that appear in practice can be easily made layered, and in fact layering makes the BDD at most quadratically larger [14]. Quadratic increase in computation time is not desirable in the case of CPIR. We will now show that one can make $\mathsf{WP}(f)$ layered in a virtually costless way. For this, first one has to add $m - d - j$ dummy nodes per each node on bottom d levels, or

$$\sum_{j=1}^{m-d} (m-d-j)(2^{2^j} - 2^{2^{j-1}}) = \sum_{j=1}^{m-d-1} 2^{2^j} + 2 = \Theta(2^{2^{m-d-1}})$$

nodes in total. Now, if d is chosen according to Eq. (1), this will be $(2 + o(1))2^{m/2}/m$ nodes. If d is chosen according to Eq. (2), this will be $(\sqrt{2} + o(1))2^{m/2}/\sqrt{m}$ nodes. Both values are negligible compared to the total number of nodes $\Theta(2^m/m)$ in the BDD. In addition, for every node on the level d—and there are $(1 + o(1))2^m/m^2$ such nodes if d is chosen according to Eq. (1) and $(1 + o(1))2^m/m$ if according to Eq. (2)—there are database-dependent edges to nodes in bottom layers. Because the number of bottom layers is $m - d$, if d

is chosen according to Eq. (2), at most $(1+o(1))2^m \log m/m^2$ new edges will be added.

After making the BDD layered, we must add privacy against a malicious server. There are many existing CPIR-to-OT transformations. In particular, the transformation from [16] takes $m = \log_2 n$ public-key operations, and modifies the CPIR protocol to a server-private protocol. (With the caveat that the public key has to be rough.) See [16,14] for more discussions.

Balancing. Let $f : \{0,1\}^m \to \{0,1\}^{\sigma\ell}$. In PrivateBDD, the client sends m messages and the server sends σ messages. If $m \gg \sigma$ and $\sigma\ell \gg m$, then one can improve the communication complexity by balancing, as follows. Without loss of generality, assume that $\sigma \mid m$. Denote $b := m/\sigma$. Then, for $j \in \{0,\ldots,b-1\}$, let $f_j : \{0,1\}^m \to \{0,1\}^{\sigma\ell/b}$. Here, f_0 computes the first ℓ/b bits of every source (that is, computes f restricted on the first ℓ/b bits of the sink values), f_1 computes the next ℓ/b bits of every source, etc. We then execute the PrivateBDD protocol (by reusing client's first message) in parallel for every f_j, and concatenate the private outputs. Thus, in this balanced version, the client sends m messages of length $\leq (\ell/b + (\mathsf{len}(\mathcal{F})+2)\cdot k)$. The server returns $b\sigma$ messages of the same length. Thus, the total communication complexity of the balanced protocol is $\leq (m+b\sigma)(\ell/b + (\mathsf{len}(\mathcal{F})+2)\cdot k) = 2m(\sigma\ell/m + (\mathsf{len}(\mathcal{F})+2)\cdot k) = 2\sigma\ell + 2m(\mathsf{len}(\mathcal{F})+2)\cdot k$. Thus, if $\ell \gg m\cdot\mathsf{len}(\mathcal{F})\cdot k$, then this version of the PrivateBDD protocol has information rate $1/2$.

If ℓ is even longer, then one can define $b := \alpha m/\sigma$ for some $\alpha > 1$, then the balanced protocol has communication complexity $(1+1/\alpha)\sigma\ell + (\alpha+1)m(\mathsf{len}(\mathcal{F})+2)k$, or—for large values of ℓ and α—information rate $1+o(1)$. For example, one can take $\alpha = \log_2 m$, then the communication complexity is $(1+o(1))\sigma\ell + (\log_2 m+1)\cdot m\cdot(\mathsf{len}(\mathcal{F})+2)k$.

In another variant of balancing, we define $\ell_{\max} \approx (\ell + \mathsf{len}(\mathcal{F})k)/b$. Then, after every b levels of the BDD, the length of the intermediate output values grows longer than $\ell_{\max}$, which requires us to double the remaining of the BDD like say in [23]. Here, the total communication complexity is $(m+2^b\sigma)/b\cdot(\ell+\mathsf{len}(\mathcal{F})\cdot k)$. Defining $b := \log_2(m/\sigma)$, this will become $(m+2)/(\log_2 m - \log_2\sigma)\cdot(\ell+\mathsf{len}(\mathcal{F})\cdot k)$. For integer b, the communication complexity is $(1+o(1))\cdot m/(\log_2 m - \log_2\sigma)\cdot(\ell+\mathsf{len}(\mathcal{F})\cdot k)$.

By using the first balancing technique, the communication complexity of Lipmaa's $(n,1)$-CPIR protocol from [18] can be improved to $2\ell + (1+o(1))\cdot\log_2^2 n\cdot k$. By using the second balancing technique, the communication complexity of Lipmaa's $(n,1)$-CPIR protocol can be improved to $\Theta((\ell\cdot\log n + k\cdot\log^2 n)/\log\log n)$. Balancing can also be used on some variations of the BddCpir protocol, especially when ℓ is large.

More Optimizations. Note that in the case of Lipmaa's $(n,1)$-CPIR protocol, the client knows in advance in what depth every of the BDD input variable x_j is used. Thus, he does not have to send values $\mathsf{Q}(\ell_{\max}, x_j)$, but can send values $\mathsf{Q}(\ell + (j-1)\cdot k, x_j)$ for every j. This optimization was used in [18] but it is

also valid in other similar contexts, including both presented variations of the BddCpir protocol.

Further Work. The first (though somewhat sketchy) version of BddCpir was presented in an earlier preprint [20] in Spring of 2008, and it is based on a very standard security assumption. In the meantime, several fully-homomorphic public-key cryptosystems have been proposed, starting with [11]. Given a fully-homomorphic cryptosystem where on can encrypt integers modulo a large N, it is easy to construct a CPIR protocol with communication complexity $\Theta(\log n + k)$, see e.g. [19] or an earlier eprint version of [18] (as available from the author's homepage), though there might be earlier works. We are currently working on a paper that shows that based on a fully-homomorphic cryptosystem, one can construct a CPIR protocol with communication complexity $\Theta(\log n + k)$ (note that all trivial approaches result in communication complexity $\Theta(k \cdot \log n)$, since Gentry's cryptosystem makes it only possible to encrypt Boolean values) and sublinear computation. Nevertheless, it is important to achieve sublinear computation under a well-known assumption, as done in the current paper.

Acknowledgments. The author was supported by Estonian Science Foundation, grant #8058, European Union through the European Regional Development Fund and the 6th Framework Programme project AEOLUS (FP6-IST-15964).

References

1. Agrawal, R., Srikant, R.: Privacy-Preserving Data Mining. In: Proceedings of the 2000 ACM SIGMOD Conference on Management of Data. pp. 439–450. Dallas, TX, USA (May 2000)
2. Aguilar-Melchor, C., Gaborit, P.: A Lattice-Based Computationally-Efficient Private Information Retrieval Protocol. In: WEWORC 2007. pp. 50–54. Bochum, Germany (Jun 2007), http://eprint.iacr.org/2007/446
3. Beimel, A., Ishai, Y., Malkin, T.: Reducing the Servers Computation in Private Information Retrieval: PIR with Preprocessing. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 55–73. Springer-Verlag, Santa Barbara, USA (Aug 20–24, 2000)
4. Breitbart, Y., Hunt III, H.B., Rosenkrantz, D.J.: On The Size of Binary Decision Diagrams Representing Boolean Functions. Theoretical Computer Science 145(1&2), 45–69 (1995)
5. Canetti, R., Ishai, Y., Kumar, R., Reiter, M.K., Rubinfeld, R., Wright, R.N.: Selective Private Function Evaluation with Applications to Private Statistics. In: PODC 2001. pp. 293–304. ACM Press, Newport, Rhode Island, USA (Aug 26–29, 2001)
6. Carbunar, B., Sion, R.: On the Computational Practicality of Private Information Retrieval. In: NDSS 2007. pp. ?–?. San Diego, California, USA (Feb 27–Mar 2, 2007)
7. 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)

8. Damgård, I., Jurik, M.: A Length-Flexible Threshold Cryptosystem with Applications. In: Safavi-Naini, R. (ed.) ACISP 2003. LNCS, vol. 2727, pp. 350–364. Springer-Verlag, Wollongong, Australia (Jul 9-11, 2003)
9. Fujita, M., McGeer, P.C., Yang, J.C.Y.: Multi-Terminal Binary Decision Diagrams: An Efficient Data Structure for Matrix Representation. Formal Methods in System Design 10(2/3), 149–169 (1997)
10. Gasarch, W., Yerukhimovich, A.: Computationally Inexpensive cPIR (2007), work in progress, available at `http://www.cs.umd.edu/~arkady/`, as of January, 2009
11. Gentry, C.: Fully Homomorphic Encryption Using Ideal Lattices. In: Mitzenmacher, M. (ed.) STOC 2009. pp. 169–178. ACM Press, Bethesda, MD, USA (May 31–Jun 2, 2009)
12. 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 (Jul 11–15, 2005)
13. Heap, M.A., Mercer, M.R.: Least Upper Bounds on OBDD Sizes. IEEE Transactions on Computers 43(6), 764–767 (1994)
14. 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)
15. Kushilevitz, E., Ostrovsky, R.: Replication is Not Needed: Single Database, Computationally-Private Information Retrieval. In: FOCS 1997. pp. 364–373. IEEE Computer Society, Miami Beach, Florida (Oct 20–22, 1997)
16. Laur, S., Lipmaa, H.: A New Protocol for Conditional Disclosure of Secrets And Its Applications. In: Katz, J., Yung, M. (eds.) ACNS 2007. LNCS, vol. 4521, pp. 207–225. Springer-Verlag, Zhuhai, China (Jun 5–8, 2007)
17. Liaw, H.T., Lin, C.S.: On the OBDD-Representation of General Boolean Functions. IEEE Transactions on Computers 41(6), 661–664 (1992)
18. 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)
19. Lipmaa, H.: New Communication-Efficient Oblivious Transfer Protocols Based on Pairings. In: Wu, T.C., Lei, C.L., Rijmen, V., Lee, D.T. (eds.) ISC 2008. LNCS, vol. 5222, pp. 441–454. Springer-Verlag, Taipei, Taiwan (Sep 15–18, 2008)
20. Lipmaa, H.: Private Branching Programs: On Communication-Efficient Cryptocomputing. Tech. Rep. 2008/107, International Association for Cryptologic Research (2008), available at http://eprint.iacr.org/2008/107
21. Naor, M., Pinkas, B.: Oblivious Transfer And Polynomial Evaluation. In: STOC 1999. pp. 245–254. ACM Press, Atlanta, Georgia, USA (May 1–4, 1999)
22. 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)
23. Stern, J.P.: A New And Efficient All Or Nothing Disclosure of Secrets Protocol. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 357–371. Springer-Verlag, Beijing, China (Oct 18–22, 1998)
24. Wegener, I.: Branching Programs and Binary Decision Diagrams: Theory and Applications. Monographs on Discrete Mathematics and Applications, Society for Industrial Mathematics (2000)