Private Branching Programs: On Communication-Efficient Cryptocomputing

Fourth eprint version, September 30, 2008

Helger Lipmaa

Cybernetica AS, Estonia, `http://research.cyber.ee/~lipmaa`

Abstract We polish a recent cryptocomputing method of Ishai and Paskin from TCC 2007. More precisely, we show that every function can be cryptocomputed in communication, linear in the product of client's input length and the length of the branching program, and computation, linear in the size of the branching program that computes it. The method is based on the existence of a communication-efficient $(2, 1)$-CPIR protocol. We give several nontrivial applications, including: (a) improvement on the communication of Lipmaa's CPIR protocol, (b) a CPIR protocol with log-squared communication and sublinear server-computation by giving a secure function evaluation protocol for Boolean functions with similar performance, (c) a protocol for PIR-writing with low amortized complexity, (d) a selective private function evaluation (SPFE) protocol. We detail one application of SPFE that makes it possible to compute how similar is client's input to an element in server's database, without revealing any information to the server. For SPFE, we design a 4-message extension of the basic protocol that is efficient for a large class of functionalities.

Keywords. Branching program, computationally-private information retrieval, cryptocomputing.

1 Introduction

A branching program [Weg00], see Sect. 2, is a fanout-2 directed acyclic graph where the internal 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 source value of the branching program on some assignment a_i of the variables is recursively computed as follows. Computation starts from this source and follows the outgoing edge that is labeled by the assignment of source's label. The same process is repeated until the computation reaches some sink. That sink value is then also the value of the corresponding source. The value $f(a)$ of the function at $a = (a_0, \dots, a_{m-1})$ is equal to the concatenation of the values of the σ sources; thus a branching program computes a function $f : \{0,1\}^m \to \{0,1\}^{\sigma\ell}$.

An alternative and less efficient way of evaluating the branching program starts from the sinks and ends with the sources. Let be v be some node (labeled by x_i), such that the values R_{v_i} of the end nodes of the outgoing a_i-labeled edges are known but the value R_v is not yet known. Then one sets $\mathsf{R}_v \leftarrow \mathsf{R}_{v_i} = (\mathsf{R}_{v_0}, \mathsf{R}_{v_1})[i]$. Thus, every node can be seen as implementing a $(2, 1)$-selector/branch operation. Recently, Ishai and Paskin [IP07] proposed a protocol for cryptocomputing (that is, computing on ciphertexts) any function f by first fixing the input length m and then designing an efficient branching program P_f for this length.[1] Their protocol uses a private version of the alternative way of computing a branching program. There, all computations are done on "ciphertexts", and the local selector/branch operation $\mathsf{R}_v = (\mathsf{R}_{v_0}, \mathsf{R}_{v_1})[i]$ is implemented by using a communication-efficient two-message $(2, 1)$-CPIR protocol. More precisely, for

[1] The $(n, 1)$-CPIR protocols by [KO97,Ste98,Lip05] use exactly the same idea to implement a private version of a complete n'-ary (multi-terminal) ordered decision tree for some $n' \ll n$, though also this was probably first explicitly stated in [IP07]. Recall that in an $(n, 1)$-computationally-private information retrieval (CPIR) protocol [CGKS95], the client has a query $x \in \{0, \dots, n-1\}$ and the server has a database $f = (f_0, \dots, f_{n-1})$ of ℓ-bit strings for some ℓ. At the end of the protocol, client obtains f_x while the server remains completely clueless about the value of x.

client's every input variable x_i to f, the client sends to the server the first message Q_i of the underlying $(2,1)$-CPIR protocol. After that, the server recursively computes a value R_v of every non-sink node of P_f as explained earlier. However, R_v is going to be equal to the second message of the $(2,1)$-CPIR protocol that uses Q_i and the database $(\mathsf{R}_{v_0}, \mathsf{R}_{v_1})$. (For a sink v, R_v is just equal to its label.) At the end, the server sends to the client R_v for every source v, and the client applies recursively the local decoding procedure to each such R_v to obtain the value of the branching program. We call this protocol PrivateBP. See Sect. 3 for a precise description of the PrivateBP protocol.

Clearly, the PrivateBP protocol is client-private because the server only sees first messages of m different $(2,1)$-CPIR protocols. Importantly, the PrivateBP protocol reveals to the client only the length—but not the shape or even the size—of the branching program. This makes it possible to use the PrivateBP to solve problems where client's input is some m-bit string and server's input is a function $f : \{0,1\}^m \to \{0,1\}^{\sigma\ell}$ from some set $\mathcal{F}$, such that all functions from $\mathcal{F}$ can be computed by branching programs of polynomial size. This restricts the size of branching program only *after* fixing server's input.

Our Contributions. While the PrivateBP protocol was proposed in [IP07], the authors did only propose one, though very interesting, application of it in keyword search. We aim to popularize the PrivateBP protocol by solving a few well-known open problems in cryptographic protocol design that are interesting in their own right. This shows that PrivateBP is also useful in practice. In particular, almost direct applications of the PrivateBP protocol—combined with results from the theory of branching programs—result in protocols that are more communication-efficient than previously known protocols for the same tasks. This phenomenon is not common say with Yao's protocol [Yao82] that is mostly much less (communication-)efficient than specialized protocols. See Sect. 3 for comparison.

Optimized PrivateBP *Protocol.* Assume that for every function $f \in \mathcal{F}$ there exists a branching program P_f with size $\mathsf{size}(P_f)$ and length $\mathsf{len}(P_f)$ for computing $f(x)$. Define $\mathsf{len}(\mathcal{F}) := \max_{f\in\mathcal{F}} \mathsf{len}(P_f)$. (For server's privacy, it is additionally required for P_f to be layered.) Let k be the security parameter. Currently, Lipmaa's $(2,1)$-CPIR protocol [Lip05] is the most efficient known $(2,1)$-CPIR protocol for the purpose of the PrivateBP protocol. As we show, by using this underlying protocol one can cryptocompute $f(x)$ with communication and client's computation $\Theta((m+\sigma)(\ell + \mathsf{len}(\mathcal{F})\cdot k))$, and server's computation $\tilde{\Theta}(\mathsf{size}(P_f))$. Then, by proposing two different balancing techniques, we get slightly better communication $\Theta((m/\log(m/\sigma))(\ell + \mathsf{len}(\mathcal{F})k))$ and alternatively, $(1+o(1))\sigma\ell + \Theta(m\cdot\mathsf{len}(\mathcal{F}))\cdot k$. See Sect. 3. Thus, the PrivateBP protocol is even more efficient than claimed in [IP07] and in particular, for *large* values of ℓ it achieves optimal communication $(1+o(1))\sigma\ell$ while a non-private protocol requires at least $\sigma\ell$ bits of communication.

Computation-Efficient CPIR. It is well-known that computation-efficiency is the main bottleneck in deploying $(n,1)$-CPIR in practice: because the online computation complexity of existing CPIR protocols is $\Omega(n)$ public-key operations, one is usually restricted to databases of size say $n = 2^{16}$ or even $n = 2^{12}$. This has motivated quite some attention on this aspect of the CPIR protocols, see for example [CS07]. In all previous protocols, the server needs to do $\Omega(n)$ online operations and it has been a long-standing open problem to prove or disprove that this is also a lower bound for sublinear-communication CPIR protocols.

We solve this problem in the case $\ell = 1$. Namely, we construct a $(n,1)$-CPIR protocol where the online computational complexity of the server depends heavily on the concrete database itself and is upperbounded by $O(n/\log n)$ online public-key operations in the worst case. (The bit-cost of a public-key operation depends on the position of a node in the branching program, see Sect. 4.1.) In this protocol we write down a branching program for the function f, $f(x) := f_x$ and then use the PrivateBP protocol to make it private. Thus, the upperbound $\Theta(n/\log n)$ follows from the upperbound on the size of ordered binary decision diagrams to compute an arbitrary Boolean function [Sha49,BHR95]. This upperbound is also tight for all but an exponentially small fraction of databases, that is, Boolean functions [Weg00]. The offline computational complexity is also $O(n/\log n)$ while the communication is $\Theta(\log^2 n)$. Altenratively, the same result shows that one can implement secure function evaluation of any $f : \{0,1\}^m \to \{0,1\}$ with communication $O(m^2)$ and computation $O(2^m/m)$. See Sect. 4.1 for how to protect server's privacy.

PIR-Writing. We also study PIR-writing where on client's private index x and a private secret y, the server updates the xth element of his database to y. It is assumed that the database is encrypted, so that the server does not know any elements of the database, nor which database element was updated. Let n be the database size. The first non-trivial solution to this problem, with $\Theta(\sqrt{n})$ communication, was proposed in [BKOS07]. Their solution was based on bilinear maps. We propose second non-trivial solution that, for some upper bound u on the number of updates, achieves *amortized* communication complexity $\Theta(u^2 \log^2 n)$. This improves upon the protocol of [BKOS07] for $u = o(\sqrt[4]{n}/\log n)$. However, as this is only the second non-trivial PIR-writing protocol, we hope it will point to a new and interesting research direction.

Selective Private Function Evaluation (SPFE). In (n, n')-SPFE [CIK$^+$01], the client submits a tuple of private indexes $x = (x_1, \dots, x_{n'})$ and a private secret y to the server, who replies with the value $f(f_{x_1}, \dots, f_{x_{n'}}, y)$ of a predetermined function f on y and server's database x_jth elements. We show that by using the PrivateBP protocol one can implement a large class of functionalities f in polynomial-time and log-squared communication. Our application to the case $n' = 1$ is almost straightforward. In the case $n' > 1$, however, we significantly extend the PrivateBP protocol; our extensions make use of the concrete properties of Lipmaa's $(2,1)$-CPIR protocol and result in 4-message protocols. (Thus, strictly speaking, they are not anymore cryptocomputing protocols.)

Our results can for example be applied in the next setting in biometric authentication that is known as fuzzy private matching [CH08]. The client of the SPFE protocol collects a fingerprint j of some person, together with her claim that she is the xth employee. Then the client contacts a server who stores encrypted fingerprint templates of all employees. At the end of the protocol, the client gets to know that person's fingerprint is sufficiently close to the xth fingerprint template in database to warrant access, without getting to know anything else. On the other hand, the server does not get to know her fingerprint, or the value of x. Assume that two fingerprints, represented as Boolean vectors of dimension ℓ "match" if at least t of their coordinates match. By using the branching program for threshold function proposed in [ST97] and the CPIR of [GR05], we get a fuzzy private matching program with communication $\Theta(\ell^2 \log^2 \ell/\log\log\ell \log\log\log\ell + \log n)k$ and server-side computation $\Theta(n \cdot \ell \log^3 \ell/\log\log\ell \log\log\log\ell)$.

Other applications. We mention a few other applications. For example, by using the PrivateBP one can solve both Yao's millionaire's problem [Yao82] and the secure vector dominance problem for m-bit vectors with computation $\Theta(m)$ and communication $(m+1)(m+2)k = \Theta(m^2)k$. The proposed generic balancing techniques make it possible to improve the communication of Lipmaa's $(n,1)$-CPIR protocol from $\Theta(\ell \cdot \log n + k \cdot \log^2 n)$ to $\ell + \Theta(k \cdot \log^2 n)$ or to $\Theta((\ell \cdot \log n + k \cdot \log^2 n)/\log\log n)$.

Many cryptographic protocols are based on the use of (additively) homomorphic encryption and because of that, are limited to cryptocomputing affine functions. Using bilinear-map based cryptosystems [BGN05] makes it possible to cryptocompute quadratic functions. The power of both types of cryptosystems is rather well-understood. For example, it was recently proved in [OS08] that when purely based on a homomorphic cryptosystem (resp., cryptosystem from [BGN05], a PIR-writing protocol has communication lower bound $\Theta(n)$ (resp., $\Theta(\sqrt{n})$). Because Lipmaa's $(2,1)$-CPIR protocol is based on a length-flexible homomorphic cryptosystem [DJ01], the PrivateBP protocol also shows that when using such a cryptosystem, one can cryptocompute a much larger class of functions. In fact, the PrivateBP protocol uses only one application of such cryptosystems (Lipmaa's $(2, 1)$-CPIR protocol) while the precise computational power of length-flexible cryptosystems may actually be even larger.

2 Preliminaries, Related Work, Cryptographic Tools

Notation. m denotes the length of client's input x. Server's input is a function $f : \{0, 1\}^m \rightarrow \{0, 1\}^{\sigma\ell}$ for suitably chosen σ and ℓ. We also denote $f(x)$ by f_x, that is, we let f to be the characteristic function of a vector $f = (f_0, \dots, f_{2^m-1})$. In CPIR-like applications, n denotes the database size. All logarithms have base 2. Finally, k is always the security parameter.

Branching Programs. A *branching program* [Weg00] is a fanout-2 directed acyclic graph where the internal 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. Every source and every assignment of the variables corresponds to 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 branching program. A branching program that has σ sources computes some function $f : \{0, 1\}^m \rightarrow \{0, 1\}^{\sigma\ell}$; $\sigma = \ell = 1$ by default.

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 branching program is a *decision tree* if the underlying graph is a tree. A branching program 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 branching program P let $\mathsf{len}(P)$ be its length (that is, the length of its longest path), $\mathsf{size}(P)$ be its size, and $\mathsf{size}_{int}(P)$ be its size without counting in the sinks. Denote by $\mathsf{BP}(f)$/$\mathsf{OBDD}(f)$ the minimal size of any branching program/OBDD computing f. Clearly $\mathsf{BP}(f) \leq \mathsf{OBDD}(f)$. It is known that any Boolean function $f : \{0, 1\}^m \rightarrow \{0, 1\}$ has $\mathsf{OBDD}(f) \leq (3 + o(1))2^m/m$ [Sha49]; for $m \geq 16$ this can be improved to $\mathsf{OBDD}(f) \leq (2+o(1))2^m/m$ [BHR95] though in practice even then Shannon's construction is as efficient. For $m \geq 25$, [BHR95] proved a precise upperbound $\mathsf{BP}(f) \leq (1 + o(1))2^m/m$. The latter upperbound is also tight because for all but an exponentially small fraction of $2^{-2^{m/2}}$ of Boolean functions the size of the best branching program is $(1 - m^{-1/2})2^m/m = (1 - o(1))2^m/m$ [Weg00, Thm. 2.2.2]. A language has a polynomial-size branching program if and only if it belongs to the complexity class $\mathbf{L/poly}$ [Cob66], that is, if it can be decided by a nonuniform log-space Turing machine. See [Weg00] for more.

Public-Key Cryptosystems. Let $\mathsf{P} = (\mathsf{G}, \mathsf{E}, \mathsf{D})$ be a length-flexible additively-homomorphic public-key cryptosystem [DJ01], where as always G is a randomized key generation algorithm, E is a randomized encryption algorithm and D is a decryption algorithm. In the case of the DJ01 cryptosystem from [DJ01], in particular, for every integer $s > 0$, $\mathsf{E}^s_{\mathsf{pk}}(\cdot)$ is a valid plaintext of $\mathsf{E}^{s+1}_{\mathsf{pk}}(\cdot)$. More precisely, $|\mathsf{E}^s_{\mathsf{pk}}(\cdot)| = sk$. (In some other length-flexible cryptosystems [DJ03], the resulting ciphertext is longer.) Recall that in a length-flexible additively-homomorphic cryptosystem, $\mathsf{E}^s_{\mathsf{pk}}(M_1) \cdot \mathsf{E}^s_{\mathsf{pk}}(M_2) = \mathsf{E}^s_{\mathsf{pk}}(M_1 + M_2)$, and moreover, $\mathsf{E}^s_{\mathsf{pk}}(M)$ is a valid plaintext of $\mathsf{E}^{s+1}_{\mathsf{pk}}(\cdot)$, so that one can legally multiple-encrypt messages as say in $\mathsf{E}^{s+2}_{\mathsf{pk}}(\mathsf{E}^{s+1}_{\mathsf{pk}}(\mathsf{E}^s_{\mathsf{pk}}(M)))$. We will explicitly need the existence of a compression function C that, given pk, s' and s for $s' \geq s$, and $\mathsf{E}^{s'}_{\mathsf{pk}}(M)$ for M a valid plaintext of $\mathsf{E}^s_{\mathsf{pk}}(\cdot)$, returns $\mathsf{E}^s_{\mathsf{pk}}(M)$. As shown in [Lip05], DJ01 has a very simple compress function.

In the CPA (*chosen-plaintext attack*) game, the challenger first generates a random $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{G}$, and sends pk to the attacker. Attacker chooses two messages M_0, M_1 and a length parameter s, and sends them to the challenger. Challenger picks a random bit b, and sends a ciphertext $\mathsf{E}^s_{\mathsf{pk}}(M_b)$ to attacker. Attacker outputs a bit b', and wins if $b = b'$. A cryptosystem is *CPA-secure* if the probability that any polynomial-time attacker wins in the CPA-game is negligibly different from $1/2$. Now, because of the existence of the compress function, a CPA-secure length-flexible cryptosystem remains CPA-secure also when the challenger can send many message pairs (M_{j0}, M_{j1}) and length parameters s_j, and has to guess b after seeing encryptions of all M_{jb} under the corresponding length parameters s_j. This so-called LFCPA-security [Lip05] of the cryptosystem is crucial for the efficient $\mathsf{PrivateBP}$ protocol as defined in the next section.[2] The DJ01 cryptosystem [DJ01] is CPA-secure under the Decisional Composite Residuosity Assumption [Pai99].

Computationally-Private Information Retrieval. In a 1-out-of-n computationally-private information retrieval protocol, $(n, 1)$-CPIR, for ℓ-bit strings, the client has an index $x \in \{0, \dots, n-1\}$ and the server has a database $f = (f_0, \dots, f_{n-1})$. The client obtains f_x. In the following we need a CPIR protocol that has the next property. We say a CPIR protocol Γ is $\mathsf{PrivateBP}$-friendly if it satisfies the next three 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. There exists a compress function C that maps $\mathsf{Q}(\ell', x)$ to $\mathsf{Q}(\ell, x)$ for any $\ell' \geq \ell$ and x.

More formally, $\Gamma = (\mathsf{Q}, \mathsf{R}, \mathsf{A}, \mathsf{C})$ 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 $\ell' \geq \ell$ and x. If the existence of C is not required we also write $\Gamma = (\mathsf{Q}, \mathsf{R}, \mathsf{A})$ even if C exists.

Lipmaa's $\mathsf{PrivateBP}$-friendly $(2, 1)$-CPIR Protocol [Lip05]. Let $\mathsf{P} = (\mathsf{G}, \mathsf{E}, \mathsf{D})$ be a length-flexible additively homomorphic public-key cryptosystem. Let s be an integer such that $sk \geq \ell \geq (s-1)k$, that is, $s \leftarrow \lceil \ell/k \rceil$. Client's private input is $x \in \{0, 1\}$, server's private input is $f = (f_0, f_1)$ for $f_0, f_1 \in \{0, 1\}^\ell$. The protocol consists of the next two steps:

1. Client generates a new key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{G}$. He sets $\mathsf{Q}_1 \leftarrow \mathsf{pk}$, $\mathsf{Q}_2 \leftarrow \mathsf{E}^s_{\mathsf{pk}}(x)$. He sends $\mathsf{Q}(\ell, x) \leftarrow (\mathsf{Q}_1, \mathsf{Q}_2)$ to server.
2. Server replies with $\mathsf{R}(\ell, f, (\mathsf{Q}_1, \mathsf{Q}_2)) \leftarrow \mathsf{E}^s_{\mathsf{Q}_1}(f_0) \cdot \mathsf{Q}_2^{f_1 - f_0}$.

If $x \in \{0, 1\}$, then clearly $\mathsf{R}(\ell, f, \mathsf{Q}) = \mathsf{E}^s_{\mathsf{pk}}(f_0 + (f_1 - f_0) \cdot x) = \mathsf{E}^s_{\mathsf{pk}}(f_x)$. Thus, it is sufficient for the client just to decrypt R. If P has a compress function, then also Lipmaa's $(2, 1)$-CPIR protocol has a compress function C. By using a hybrid argument it is easy to show that is safe for the client to send polynomially many queries $\mathsf{E}^{s_i}_{\mathsf{pk}}(M_i)$ encrypted by using the same public key pk. The important properties of this $\mathsf{PrivateBP}$-friendly $(2, 1)$-CPIR protocol that make it efficient in our applications are that $|\mathsf{Q}(\ell, \cdot)| = sk + k \leq \ell + 2k$ and $|\mathsf{R}(\ell, \cdot, \cdot)| = sk + k \leq \ell + 2k$.

Cryptocomputing protocols. Similarly to a CPIR protocol, a (two-message) cryptocomputing protocol is a triple $(\mathsf{Q}, \mathsf{R}, \mathsf{A})$ of efficient protocols, where client has an input $x = (x_0, \dots, x_{m-1})$ with $x_j \in \{0, 1\}$, server has an input $f : \{0, 1\}^m \to \{0, 1\}^\ell$. The client starts the protocol by sending $\mathsf{Q} := \mathsf{Q}(\ell, x)$ to the server, who replies by sending $\mathsf{R} := \mathsf{R}(\ell, f, \mathsf{Q})$ to the client. Finally, the client (who like always is assumed to be stateful) outputs $\mathsf{A}(\ell, x, \mathsf{R})$ as his private output.

Semisimulatable Security. Let $\Gamma = (\mathsf{Q}, \mathsf{R}, \mathsf{A})$ be any (two-message) cryptocomputing protocol between a client and a server. Within this work we use the convention of many previous papers on oblivious transfer [AIR01,Lip05] that only require privacy in the malicious model. Various papers, for example [FIPR05,IP07], also recommend to use this model for other cryptographic protocols. More precisely,

[2] It seems that the relevance of the compress function to the equivalence of CPA and LFCPA-security has not been pointed out before.

client's privacy is guaranteed in the sense of indistinguishability (CPA-security), while server's privacy is guaranteed in the sense of simulatability. This assumption makes it possible to design two-message cryptocomputing protocols that are both communication and computation-efficient. We now give an informal definition of semisimulatability for any cryptocomputing protocol, following many earlier papers like [NP99,AIR01,FIPR05,Lip05,IP07].

For the *CPA-security* (that is, the privacy) *of the client*, a malicious nonuniform probabilistic polynomial-time server is required not to be able to distinguish between client's messages $\mathsf{Q}(\ell, x_0)$ and $\mathsf{Q}(\ell, x_1)$ corresponding to any two of client's inputs x_0 and x_1. For *server-privacy*, we require the existence of a 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 *semisimulatably secure* if it is both client-private and server-private.

Any CPIR protocol Γ is required to be client-private, that is, CPA-secure. Lipmaa's $(2,1)$-CPIR protocol [Lip05], when based on the DJ01 cryptosystem [DJ01], is CPA-secure under the Decisional Composite Residuosity Assumption [Pai99]. Because of the existence of the C 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 $\{\mathsf{Q}(\ell_i, x_{i1}))\}$. A semisimulatable $(n,1)$-CPIR protocol is also known as an $(n,1)$-*oblivious transfer protocol*.

3 The PrivateBP Protocol

Next, we describe the PrivateBP (*private branching programs*) cryptocomputing protocol from [IP07]. It generalizes the cryptocomputing process, done in several previous CPIR protocols [KO97,Ste98,Lip05]. Our exposition is simpler than the more general exposition of [IP07] with almost straightforward security proofs. The concrete protocol has also some small differences compared to the protocol of [IP07].

In the PrivateBP protocol, the client has private input $x \in \{0,1\}^m$, the server has private input $f : \{0,1\}^m \to \{0,1\}^{\sigma\ell}$ 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 a branching program P_f that all have σ sources and ℓ-bit sink labels. Let $\mathsf{len}(\mathcal{F}) := \max_{f\in\mathcal{F}} \mathsf{len}(P_f)$. Let $\Gamma' = (\mathsf{Q}', \mathsf{R}', \mathsf{A}', \mathsf{C}')$ be a PrivateBP-friendly $(2,1)$-CPIR protocol. Define $|\mathsf{Q}^{(1)}(\ell)| := |\mathsf{Q}'(\ell, x)|$, $|\mathsf{R}^{(j)}(\ell)| := |\mathsf{R}'(|\mathsf{Q}^{(j)}(\ell)|, x, \mathsf{Q}')|$ and $|\mathsf{Q}^{(j+1)}(\ell)| := |\mathsf{Q}'(|\mathsf{R}^{j}(\ell)|, x)|$. We assume that those values are well-defined, that is, they do not depend on the concrete values of x and f; because Γ' has to be secure, this assumption is reasonable.

Description of PrivateBP**.** For a fixed f, the server executes the branching program P_f bottom-up, that is, from the sinks to the source. The output values R_v of the sinks are equal to their labels. At every node v of the branching program with label x_v 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 $\mathsf{R}_{v_{x_v}}$ upwards as R_v. The server does this for all nodes in some order, and then sends the output values of the σ sources to the client. For every source, the client applies the decoding procedure A' repeatedly to obtain the label of the unique sink that is uniquely determined by this node and by client's input x. A complete description of the PrivateBP protocol for $\mathcal{F}$ is depicted by Fig. 1.

Theorem 1. *Let $\Gamma' = (\mathsf{Q}', \mathsf{R}', \mathsf{A}', \mathsf{C}')$ be a CPA-secure* PrivateBP*-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 branching program P_f. Then $\mathcal{F}$ has a CPA-secure cryptocomputing protocol with communication $m \cdot |\mathsf{Q}^{(\mathsf{len}(\mathcal{F}))}(\ell)| + \sigma \cdot |\mathsf{R}^{(\mathsf{len}(\mathcal{F}))}(\ell)|$ and online computation of $\mathsf{size}_{int}(P_f)$ $\mathsf{R}'(\ell^*, \cdot, \cdot)$ functions. In particular if P_f is layered then the protocol is server-private in the semihonest model.*

1. **Inputs:** server knows a function $f : \{0,1\}^m \to \{0,1\}^{\sigma\ell}$ from $\mathcal{F}$ and client knows $x \in \{0,1\}^m$.
2. **Offline phase:** server computes an efficient branching program P_f for f that has σ sources and has ℓ-bit sink labels; the client knows $\mathsf{len}(\mathcal{F})$. Let $\ell_{\max} := |\mathsf{Q}^{(\mathsf{len}(\mathcal{F})-1)}(\ell)|$.
3. **Online phase:**
   (a) **Client does:** For $j \in \{0,\dots,m-1\}$, set $\mathsf{Q}_j \leftarrow \mathsf{Q}'(\ell_{\max}, x_j)$. Send $\mathsf{Q}(\ell, x) \leftarrow (\mathsf{Q}_0,\dots,\mathsf{Q}_{m-1})$ to the server.
   (b) **Server does:**
      i. For sinks v of P_f set R_v to be their label, for other nodes v set $\mathsf{R}_v \leftarrow \perp$.
      ii. Do by following some ordering of the nodes:
         A. Let v be some node with $\mathsf{R}_v = \perp$ with children v_0 and v_1 that have $\mathsf{R}_{v_0}, \mathsf{R}_{v_1} \neq \perp$; 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 branching program is layered then $|\mathsf{R}_{v_0}| = |\mathsf{R}_{v_1}|$.
      iii. For any source v: send R_v to the client.
   (c) For any source v: Client computes her private output from R_v by applying A' recursively $\mathsf{len}(\mathcal{F})$ times.

Figure1. The PrivateBP protocol

Proof. The CPA-security follows straightforwardly from the CPA-security of Γ'. If P_f is layered, then the client is completely oblivious of the shape of the branching program, except the length of it: he just forms queries corresponding to his input bits by using his knowledge of the length of the branching program (and on the output length ℓ), and then receives multiple-"encryptions" of the outputs. Client's communication is $m \cdot |\mathsf{Q}^{(\mathsf{len}(\mathcal{F}))}(\ell)|$ bits, server's communication is $\sigma \cdot |\mathsf{R}^{(\mathsf{len}(\mathcal{F}))}(\ell)|$ bits. Server has to do some work per every node of P_f. □

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 $i \le \mathsf{len}(P) - 1$. This can increase the communication by a factor of $\mathsf{len}(P)$. C makes it possible to compute $\mathsf{Q}(|\mathsf{Q}^{(i)}(\ell)|, x_j)$ from $\mathsf{Q}(\ell_{\max}, x_j)$. (Note that the CPIR protocols like [Lip05] do not explicitly need the C function because they cryptocompute the ordered binary decision tree where every x_i is only tested on the ith level of the tree.)

We will assume throughout this paper that we are working with Lipmaa's underlying $(2,1)$-CPIR, see Sect. 2; this protocol is currently the most efficient $(2,1)$-CPIR for our purposes. It is the only known protocol that currently allows the PrivateBP protocol to achieve communication that is polynomial in $\mathsf{len}(\mathcal{F})$. A precise result follows:

Corollary 1. *Assume that the Decisional Composite Residuosity Assumption is true [Pai99]. Let $\mathcal{F}$ be a set of functions $f : \{0,1\}^m \to \{0,1\}^{\sigma\ell}$, and let P_f be some σ-source branching program with ℓ-bit sink labels that computes f. Then $\mathcal{F}$ has a CPA-secure cryptocomputing protocol with communication upperbounded by $k + (m+\sigma)(\ell + (\mathsf{len}(\mathcal{F}) + 2)k)$, and computation of* $\mathsf{size}(\mathcal{F})$ $\mathsf{R}'(\ell^*, \cdot, \cdot)$ *functions.*

Proof. Let $\mathsf{P} = (\mathsf{G}, \mathsf{E}, \mathsf{D})$ be the underlying length-flexible cryptosystem. This version of the PrivateBP protocol generates one single $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{G}$ and uses the same pk to construct all m queries Q_j. Because Lipmaa's $(2,1)$-CPIR is both PrivateBP-friendly and CPA-secure, the CPA-security of PrivateBP follows from a standard hybrid argument. Computation is also trivial. To calculate the communication efficiency, note that $\mathsf{Q}_j = \mathsf{Q}'(\ell_{\max}, x_j) = \mathsf{E}_{\mathsf{pk}}^{\lceil \ell/k \rceil + \mathsf{len}(\mathcal{F})}(x_j)$. Thus, $|\mathsf{Q}_j| = |\mathsf{E}_{\mathsf{pk}}^{\lceil \ell/k \rceil + \mathsf{len}(\mathcal{F})}(x_j)| = (\lceil \ell/k \rceil + \mathsf{len}(\mathcal{F}) + 1)k \le \ell + (\mathsf{len}(\mathcal{F}) + 2)k$. Thus, client sends a public key (of length say k) and at most $m \cdot (\ell + (\mathsf{len}(\mathcal{F}) + 2)k)$ additional bits. The output of the branching program is equal to σ $\mathsf{len}(\mathcal{F})$-times encryptions of sink values, where the sinks are selected by the encrypted client inputs x_j. Server's communication consists of $\sigma\ \mathsf{len}(\mathcal{F})$ times encrypted messages of length $\leq \ell + (\mathsf{len}(\mathcal{F}) + 2)k$. $\square$

In particular, if $\mathsf{len}(\mathcal{F})$ is polylogarithmic in m and $\mathsf{size}_{int}(P_f)$ is polynomial in m for every $f \in \mathcal{F}$, then also the communication is polylogarithmic in m and the computation is polynomial in m. For Boolean functions, the communication can in fact be upperbound by $k + (m+1)(\mathsf{len}(P) + 2)k$.

Stronger Security Guarantees. To guarantee server's privacy in the semihonest model, the used branching program must be layered; otherwise some information about the shape of the branching program could be leaked. (All examples in this paper in fact use layered branching programs.) In this case, a honest client will get no information except the function value. Therefore, any of the transformations of [NP99,AIR01,Kal05,Lip05,IP07,LL07] can be used to transfer the $\mathsf{PrivateBP}$ protocol from a client-private protocol to a semisimulatable protocol. Semisimulatability is a standard security notion for two-message oblivious transfer protocols, and similarly to [FIPR05,IP07] we advocate its use in general cryptocomputing. If needed, one can transform the protocols into simulatably secure protocols of sublinear communication by using the general methodology proposed in [NN01].

Balancing. In $\mathsf{PrivateBP}$, the client sends m messages and the server sends σ messages. If $m \gg \sigma$ and $\ell \gg m/\sigma$, then one can improve the communication by balancing. Without loss of generality, let $\sigma \mid m$. Denote $b := m/\sigma$. Then for $f : \{0,1\}^m \to \{0,1\}^{\sigma\ell}$, write $f(x) = (f_0(x), \dots, f_{b-1}(x))$ for $f_j : \{0,1\}^m \to \{0,1\}^{\sigma\ell/b}$. Here, f_0 computes the first ℓ/b bits of every source, etc. Apply the $\mathsf{PrivateBP}$ protocol in parallel for every f_j, and concatenate the private outputs. The client sends m messages of length $\leq (\ell/b + (\mathsf{len}(\mathcal{F}) + 2)k)$. The server sends back $b\sigma$ messages of the same length. Thus, the total communication of the balanced protocol is $\leq (m + b\sigma)(\ell/b + (\mathsf{len}(\mathcal{F}) + 2)k) = 2m(\sigma\ell/m + (\mathsf{len}(\mathcal{F}) + 2)k) = 2\sigma\ell + 2m(\mathsf{len}(\mathcal{F}) + 2)k$. For example, Lipmaa's $(n,1)$-CPIR protocol from [Lip05] has communication $\Theta(\ell \cdot \log n + k \cdot \log^2 n)$. The new balanced protocol has communication $\leq 2\ell + 2\log n(\log n + 2)k$. By redefining $b = am/\sigma$ for some large value of a, one can actually achieve communication $(1 + o(1))\ell + \Theta(\log^2 n)k$ for large ℓ.

In another variant of balancing, we define $\ell_{\max} \approx (\ell + \mathsf{len}(P)k)/b$, then the total communication becomes $\Theta(m + 2^b\sigma)(\ell + \mathsf{len}(\mathcal{F})k)/b$. Defining $b = \Theta(\log(m/\sigma))$, this will become $\Theta(m/\log(m/\sigma) \cdot (\ell + \mathsf{len}(\mathcal{F})k)$. We will not explicitly mention these balancing techniques in what follows, but many other protocols can also be made more efficient by using similar methods.

Example: $(n,1)$-CPIR. All CPIR-protocols that follow the Kushilevitz-Ostrovsky recursion technique [KO97,Ste98,Lip05] can be seen as a cryptocomputing of an ordered decision tree. (See App. A.) In particular, Lipmaa's $(n,1)$-CPIR protocol from [Lip05] uses his $(2,1)$-CPIR protocol to achieve communication $\Theta(m \cdot (\ell + \mathsf{len}(P)k)) = \Theta(\ell \cdot \log n + k \cdot \log^2 n)$, agreeing with Cor. 1. By using the balancing technique, this can be improved to either $(1+o(1))\ell + \Theta(\log^2 n \cdot k)$ or to $\Theta((\ell \cdot \log n + k \cdot \log^2 n)/\log\log n)$.

Example: Secure Vector Dominance Problem. Let the client have a vector $x = (x_0, \dots, x_{m-1})$ and the server have a vector $y = (y_0, \dots, y_{m-1})$. The client needs to know the value of a Boolean function f defined as $f(x) = 1$ if $x_j \geq y_j$ for all j, and $f(x) = 0$ otherwise. This can be done as follows. Let $\mathcal{F}$ be the set of such functions. For arbitrary $f \in \mathcal{F}$, $f(x)$ can be computed by a branching program P_f that has $\mathsf{size}_{int}(P_f) = \mathsf{len}(P_f) = m$. With the use of Lipmaa's $(2,1)$-CPIR protocol, one can thus design a protocol for secure vector dominance with communication $(m+1)(m+2)k = \Theta(m^2)k$ and server-computation $\Theta(m)$. The just presented protocol that just follows general methodology can be compared with more complex specialized protocols from say [YYWP08]. Clearly, one can also construct a $\mathsf{PrivateBP}$ protocol for Yao's millionaire's problem [Yao82] with exactly the same complexity.

Example: Cardinality Set-Intersection [KS05]. Assume that the client has a set $x \subseteq \mathbb{Z}_n$ and the server has a set $y \subseteq \mathbb{Z}_n$. The client has to compute a function $f(x) := |x \cap y|$. This can be done as follows. Let $x_j = 1$

iff $j \in x$ and $y_j = 1$ iff $j \in y$. The client's input to the protocol is $(x_0, \ldots, x_{m-1})$. The (ordered) branching program performs counting of ones in some set, see [Weg00], the set in this case is $x' = \{x_j : y_j = 1\}$. Such an OBDD has $|y|(|y|+1)/2$ internal nodes and depth $|y|$. Moreover, $\ell = \lceil \log m \rceil$. Thus the $\mathsf{PrivateBP}$ protocol in this case has communication $k+(m+\sigma)(\ell+(\mathsf{len}(\mathcal{F})+2)k) = k+(m+1)(\lceil \log m \rceil+(m+2)k) = \Theta(m^2)k$ and computation $\Theta(|y|^2) = O(m^2)$.

Optimizations. *Randomization.* Clearly, any $f \in \mathcal{F}$ can be a probabilistic function $f : \{0,1\}^m \times \mathsf{Random} \to \{0,1\}^{\sigma\ell}$ because the server can randomize the choice of P_f. *Non-Binary Branching Programs And Generalization.* One can implement n'-ary branching programs by plugging in suitably defined $(n', 1)$-CPIR protocols at the inner nodes. This does usually not improve the communication significantly. See [IP07] for more details.

Comparison to Other Cryptocomputing Protocols. By using Yao's garbled circuit approach [Yao82], one can securely compute every function g in $\mathbf{BPP/poly}$ by using communication that is linear in the circuit complexity $C(g)$ of g. In the $\mathsf{PrivateBP}$ protocol, one can cryptocompute a set of functions $\mathcal{F}$ such that for any *fixed* server's input $f \in \mathcal{F}$, $g(\cdot, f)$ belongs to a (probably) smaller class $\mathbf{L/poly}$, by using communication that is linear in the branching program *length* $\mathsf{len}(\mathcal{F})$, which is often significantly smaller than $C(f)$. Moreover, $\mathsf{PrivateBP}$ reveals only the length of $\mathcal{F}$, while Yao's protocol reveals the shape of the circuit. This may have an important practical significance: in the extreme, $\mathcal{F}$ can consist of all functions that can be computed by a branching program of fixed length.

In Yao's protocol, one has to execute $\Theta(C(g))$ private-key operations and only $\Theta(m)$ public-key operations, where m is again client's input size. On the other hand, in the $\mathsf{PrivateBP}$ protocol, one has to execute $\Theta(\mathsf{BP}(g(\cdot, f)))$ public-key operations. Because public-key operations are much more costly than private-key operations, this may limit the use of the $\mathsf{PrivateBP}$ protocol unless communication complexity is important. However, this is often not a issue because in all practical protocols $\mathsf{BP}(g(\cdot, f)) \ll C(g)$. As an example, a branching program for computing a maximum of x and fixed f, with $x, f \in \{0,1\}^m$, has size $\Theta(m)$ while a circuit to compute maximum of unfixed x and f is significantly larger.

Moreover, there are situations where communication really matters. For example, one can construct a trivial $(n, 1)$-CPIR protocol for ℓ-bit strings with communication $N \cdot \ell$ by letting the server just transfer the whole database to the client. Yao's protocol has much larger communication and computation than this trivial protocol, while the $\mathsf{PrivateBP}$ protocol, as known from [Lip05], achieves log-squared communication. Moreover, the garbled circuit protocol requires more than two rounds.

Sander, Young and Yung [SYY99] proposed a protocol for cryptocomputing everything in a (probably) smaller complexity class $\mathbf{NC^1} \subseteq \mathbf{L/poly}$ by using an arbitrary additively homomorphic cryptosystem. However, the communication in their protocol is exponential in $\mathsf{len}(f)$ while in the $\mathsf{PrivateBP}$ it is linear. Naor and Nissim [NN01] proposed another protocol that utilizes communication-complexity trees. In their protocol, assuming that the communication complexity of the nonprivate protocol is c (note that even for very simple problems $c \geq \log m'$, where m' is the summatory input length of both parties), the private version has communication complexity $O(c^2)$, computation complexity $O(c2^c)$ and round complexity that is usually larger than 2. They proposed also another protocol that utilized branching programs to achieve better computation but it also had somewhat higher communication and still (usually a much) larger number of the rounds.

4 Applications

4.1 Computation-Efficient $(n, 1)$-CPIR Protocol

Computation-efficiency is currently the main bottleneck in deploying $(n, 1)$-CPIR in practice; this has motivated quite some attention on this aspect of the CPIR protocols, see for example [CS07,GY07,AMG07]. (In the case of multiple databases, [BIM00] has also worked on sublinear-computation PIR protocols. However, the setting and the methods are completely different.) The main reason here is that in all known sublinear-communication $(n, 1)$-CPIR protocols, one has to apply at least one public-key operation per database element. This limits the size of the databases to say $n = 2^{20}$ or even to $n = 2^{14}$, depending on the computational power. We now address this question in the special case $\ell = 1$ by proposing a protocol that requires $\Theta(n/\log n)$ public-key operations in the worst case, and potentially much less work in the case of redundant databases.

Assume that $\ell = \sigma = 1$ and the database size is $n = 2^m$, that is, that the server's database consists of 2^m bits. (If n is not a power of 2 then one can round the database size up.) In this case, we can restate the $(n, 1)$-CPIR protocol as follows. Assume client has an input $x \in \{0, 1\}^m$ and the server has a Boolean function $f : \{0, 1\}^m \to \{0, 1\}$, such that $f(x) = f_x$. The client needs to retrieve $f(x)$. Thus, $\mathcal{F} = \{f : \{0, 1\}^m \to \{0, 1\}\}$. Moreover, server's work can depend heavily on f but the communication depends on $\mathsf{len}(\mathcal{F})$ so we would like to minimize the latter. Fortunately, it is well known that any Boolean function f can be computed by an OBDD P_f of size $(3 + o(1))2^m/m$ that has length m [Sha49]. *Offline* computation of such a branching program may take 2^m *non-cryptographic* operations. This value may not be so relevant because the offline computation has to be only done once per database and not per query. In fact, even if one database element changes, updating the data structure takes $\Theta(2^m/m)$ non-cryptographic operations. *Online* evaluation of the branching program on concrete input x takes thus $\Theta(2^m/m)$ operations in the worst case, and often much less. In particular, when applying the PrivateBP protocol, one needs to do $\mathsf{size}_{int}(P_f)$ online public-key operations. Because in this case $\mathsf{size}_{int}(P_f) \leq (3 + o(1))2^m/m$, the number of online public-key operations is always less than 2^m, for any possible database f. To the best of our knowledge, this is the first $(n, 1)$-CPIR with this property. (See also App. B where we describe the construction based on Shannon's upperbound.)

Note that the upperbound $\Theta(2^m/m)$ is also tight because for all but an exponentially small fraction of $2^{-2^{m/2}}$ of Boolean functions the size of the best branching program is $(1 - m^{-1/2})2^m/m = (1 - o(1))2^m/m$ [Weg00, Thm. 2.2.2]. Thus, we also have a worst-case lowerbound for the online computation of CPIR for all but an exponentially small fraction of databases. Many real-life databases fall to this exponentially small fraction due to the redundancy present in almost all such data: in fact, otherwise most of the existing data-mining and machine learning algorithms would not be useful in practice. Note that in the case of Lipmaa's $(2, 1)$-CPIR protocol, the bitcomplexity of every public-key operation (a computation of R) depends heavily on the bitlength of elements and thus on the depth of the node in the tree.

Theorem 2. *Assume that the Decisional Composite Residuosity Assumption holds. Fix n, let $m = \lceil \log n \rceil$. Then there exists a CPA-secure $(n, 1)$-CPIR protocol for 1-bit strings with communication $(m + 1)(m + 2)k = \Theta(m^2)k = \Theta(\log^2 n)k$ and online computation $O(n/\log n)$.*

Proof. Follows from Cor. 1 and Shannon's upperbound, by letting $\mathcal{F}$ to be the set of all Boolean functions $f : \{0, 1\}^m \to \{0, 1\}$. □

Longer Strings. Let $f : \{0, 1\}^m \to \{0, 1\}^\ell$ for some $\ell \geq 1$. By the upper bound of [Sha49], clearly $\mathsf{BP}(f) \leq \ell \cdot (3 + o(1))2^m/m$ by just computing ℓ branching programs in parallel. By following the proof of [Weg00], one can easily show that $\mathsf{BP}(f) \leq (3 + o(1))2^m \cdot \ell/(m + \log \ell)$. Thus, one can implement a $(n, 1)$-CPIR for ℓ bit strings with *upperbound* of $O(n \cdot \ell/(\log n + \log \ell))$ online computation. In many practical cases, the online computation is again much smaller.

Practical Relevance. If one uses the $(n, 1)$-CPIR protocols of [Lip05,GR05] for ℓ-bit strings with $\ell > m = \log n$, a more efficient way in practice would be to divide the Boolean database of size n into n/ℓ blocks of ℓ-bits and then transfer a block that contains the required bit. This would require n/ℓ public-key operations. Nevertheless, the presented protocol is still interesting for its theoretical implications and may achieve better performance if the database can be described by using a small branching program. Moreover, in several existing constructions [KO97,GY07], the authors cryptocompute an n'-ary ordered decision tree by using an underlying $(n', 1)$-CPIR protocol for 1-bit strings because no efficient CPIR for longer strings is known under assumptions like quadratic residuosity and hardness of lattice problems. In such cases, one can use our ideas to directly reduce the computation cost.

Achieving Server-Privacy. Recall that an $(n, 1)$-CPIR protocol that also achieves server-privacy is called an $(n, 1)$-OT protocol. There are many existing CPIR-to-OT transformations, already cited before. In this case, we recommend the use of the transformation from [NP99] (if one is interested in computation-efficiency) or the transformation from [NN01] (if one is interested in communication-efficiency). The latter actually achieves simulatable security, that is, a much stronger property than semisimulatable security. It is straightforward to prove that applying either of the transformations to any CPA-secure CPIR protocol results in a (semi)simulatable oblivious transfer protocol.

4.2 PIR-Writing

Assume that client has outsourced—say due to storage limitations or need to backup—his database to the server. Due to the privacy requirements, the database is encrypted. In a PIR-writing protocol, the client updates a single element of the database so that the server does not know which element was changed. That is, client has (x, y), server has an encrypted database $f = (f_0, \ldots, f_{n-1})$. Ideally, the protocol has only a single message, from the client to the server. The client has no output, while server obtains a new encrypted database f' such that f'_i and f_i decrypt to the same value if $i \neq x$, while f'_x decrypts to y.

Related Work. In a straightforward PIR-writing protocol with $\Theta(n)$ communication, the server sends the old database to the client; the client decrypts everything, updates the xth element, re-encrypts the database and sends the new database back. The next (known) approach uses an additively-homomorphic cryptosystem. The client and the server first execute a $(n, 1)$-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 encryptions of f_j with ciphertexts c_j, this clearly correctly updates the database. The first non-trivial protocol for this task was recently proposed in [BKOS07]. Essentially, a variant of it uses bilinear pairings to send $2\sqrt{n}$ 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. However, this protocol uses bilinear pairings and more than two messages. In [OS97], the authors studied the same problem in a different, information-theoretic setting with multiple databases.

New Protocol. Client's inputs are $(x_0, \ldots, x_{t-1}; y)$, where x is again client's index and y is the new value of the xth database element. Server's input is an encrypted database f with elements having (current) length ℓ. In the new PIR-writing protocol, the server runs n branching programs P_i in parallel, where P_i returns f_i if $i \neq x$, and y otherwise. Let the database of the outputs of n branching programs be f'. Instead of returning f' to the client, the server replaces f with f'. A more formal protocol description follows:

1. Client sends his query $\mathsf{Q}(\ell, i)$.

2. For $i \in \{0, \dots, n-1\}$, the server does in parallel:
   (a) Let P_i be a branching program that returns f_i if $i \neq x$, and y otherwise.
   (b) Server executes $\mathsf{PrivateBP}$ by using P_i, and sets f'_i to be equal to the resulting second message $\mathsf{R}(\ell, f, \mathsf{Q})$.
3. Server stores $f' = (f'_0, \dots, f'_{n-1})$ as the new database with elements having length $\ell' = |\mathsf{R}(\ell, \cdot, \cdot)|$.

Assume that database elements have length ℓ. Then before any updates the server stores a database with elements of size $(s+1)k$ where s is defined as usually. The first update protocol has communication complexity $\leq (t+1) \cdot (\mathsf{len}(P)k + \ell) = (t^2+t) \cdot k + (t+1)\ell$. The new database has elements of size $(s+t+1)k \leq (t+1)k+\ell$ and thus the next update protocol has communication $(t^2+t) \cdot k + (t+1)(tk+k+\ell)$. Analogously, the ith update protocol has communication $(t^2+t) \cdot k + (t+1)(itk+ik+\ell)$, and thus the first u protocols have total communication $u(t^2+t) \cdot k + (t+1)\sum_{i=0}^{u-1}(itk+ik+\ell) = O(t^2u^2k)$. Thus this protocol is more communication-efficient than the protocol of [BKOS07] if $u \leq \sqrt[4]{n}/\log n$. Thus, the new protocol has amortized communication $u^2 \cdot \log^2 n$ if the number of updates is upperbounded by u. Moreover, every update only consists of a single message from the client to the server, and the protocol does *not* use pairings. Server's computation is $O(n \log n)$.

Security. Client-privacy follows from the previous general proof. Notice that in this case we are not interested in server-privacy at all.

4.3 Selective Private Function Evaluation

Preliminaries. In [CIK$^+$01] the authors considered the problem of (n, n')-*SPFE*, where the client obtains some function $g(y, f_{x_1}, \dots, f_{x_{n'}})$ of the database elements $f_{x_1}, \dots, f_{x_{n'}}$ and client's input y. (y was not explicitly mentioned in [CIK$^+$01].) The authors proposed several ways to tackle this problem, but all such ways needed an implementation of either a secure multi-party computation protocol or Yao's garbled circuits protocol. The new protocol, described below, is significantly more efficient.

New Protocol (Case $n' = 1$). Assume that client has two inputs x and y, and server has a database $f = (f_0, \dots, f_{n-1})$. The client must obtain $g(y, f_x)$. We construct the next protocol. The client sends a Q message, corresponding to y, to the server. Server constructs n different branching programs P_{g,f_i} for computing $g(\cdot, f_i)$ for each value of $i \in [n]$. She stores the results of all those branching programs in a new database. In parallel, the client and the server execute an $(n,1)$-CPIR protocol to this database, from which the client retrieves the $x$the element.

The case $n' = 1$ was previously considered in [BCPT07], who proposed tailored protocols for a very small number of specific problems. Their approach does not seem to generalize to other problems.

Example: Comparison. As an example, suppose that the client wants to establish whether $y > f_x$, $y = f_x$ or $y < f_x$, where all values are ℓ-bit long. One can construct a branching program for this with $\mathsf{len}(P) = \mathsf{size}_{int}(P) = \ell$. If one uses the Gentry-Ramzan $(n,1)$-CPIR protocol [GR05], then the communication of this protocol is $n(\ell k + \ell) + O(\log n + \ell k + \ell + k)$, and the computation is $\Theta(n \cdot \ell)$.

Example: Fuzzy Private Matching. Suppose that the client wants to establish whether $w_h(y, f_x) < t$, where all values are ℓ-bit long and w_h denotes the Hamming distance. (See [CH08] for previous work.) This can be straightforwardly reduced to computing the threshold function $T_{\ell,t}$, where $T_{\ell,t}(v_1, \dots, v_\ell) = 1$ iff at least t bits v_i are equal to 1. The best known branching program for threshold function, with size $O(\ell \log^3 \ell / \log\log \ell \log\log\log \ell)$, though of length $O(\ell \log^2 \ell / \log\log \ell \log\log\log \ell)$, was presented in [ST97]. (It is also known from [BPRS90] that $\mathsf{BP}(f) = \Omega(\ell / \log \ell \log\log \ell)$.) Combining their solution with the Gentry-Ramzan CPIR protocol results in a fuzzy matching protocol with communication $\Theta(\ell^2 \log^2 \ell / \log\log \ell \log\log\log \ell + \log n)k$ and computation $\Theta(n \cdot \ell \log^3 \ell / \log\log \ell \log\log\log \ell)k$ which compares favorably with the “trivial protocol” but also with the best previous work [CH08]. Note that this is almost optimal communication-wise because in the non-private version the the fuzzy matching has communication $\log n + \ell + 1$ and server-side computation $\Theta(\ell + \log n)$.

General Case for Any n'. In the general case we cannot use the same technique anymore because it could result in superpolynomial computation time. Instead, we propose the next extension of the $\mathsf{PrivateBP}$ protocol that uses the fact that Lipmaa's $(2, 1)$-CPIR protocol is homomorphic, that is, $\mathsf{Q}(\ell, i_1) \cdot \mathsf{Q}(\ell, i_2) = \mathsf{Q}(\ell, i_1 + i_2)$. Description of the new (n, n')-SPFE protocol follows:

1. For $i \in [n']$ and $j \in [\lceil \log n \rceil]$, client and server execute a CPIR protocol such that the client recovers a randomized version of the jth bit $f_{x_{ij}} + r_{ij}$ of f_{x_i}. Here, r_{ij} is a fresh random bit generated by the server.
2. For $i \in [n']$ and $j \in [\lceil \log n \rceil]$, client sends $\mathsf{Q}(\ell_{\max}, f_{x_{ij}} + r_{ij})$ to the server who uses the homomorphic properties and r_{ij} to compute $\mathsf{Q}(\ell_{\max}, f_{x_{ij}})$, where again $\ell_{\max} = |\mathsf{Q}^{(\mathsf{len}(\mathcal{F})-1)}(\ell)|$ and $s_{\max} = \lceil \ell_{\max}/k \rceil$ as always.
3. Server now constructs a branching program for $g(a_1, \ldots, a_{n'}, y)$, and applies server's computation part in the $\mathsf{PrivateBP}$ protocol to it, branching on the corresponding bit of f_{x_i} instead (for which he now knows $\mathsf{Q}(\ell_{\max}(f_{x_{ij}}))$ of the corresponding bit of a_i.
4. Client recovers his private output from the outputs of the $\mathsf{PrivateBP}$ protocol.

Note that this protocol requires 4 messages, and computes $n' \cdot \log n$ $(n, 1)$-CPIR protocols and a $\mathsf{PrivateBP}$ protocol for g. Thus, when the Gentry-Ramzan CPIR [GR05] is used, it has communication $\Theta(n' \cdot \log n \cdot (\log n + \ell + k) + (m + \sigma)(\ell + \mathsf{len}(P_g)))$ and computation $\Theta(n' \cdot \log n \cdot n + \mathsf{size}(P_g))$. For most of the interesting classes $\mathcal{F}$, the proposed SPFE protocol is much more efficient than any of the solutions proposed in [CIK$^+$01].

5 Open Questions

(1). Currently, Lipmaa's $(2, 1)$-CPIR protocol is the most efficient underlying protocol. The $\mathsf{PrivateBP}$ protocol would be more communication-efficient if we had a $\mathsf{PrivateBP}$-friendly $(2, 1)$-CPIR protocol where client's first message's length did not depend on ℓ, the length of database elements. There seems to be no reason why such a $(2, 1)$-CPIR protocol would not exist. One way to achieve this is to design a length-flexible public-key cryptosystem where one can compute $\mathsf{E}^{s+1}_{\mathsf{pk}}(M)$ given only $\mathsf{E}^{s}_{\mathsf{pk}}(M)$. This would enable to get rid of the dependency from $\mathsf{len}(P)$ in communication and thus decrease total communication of the protocols from $O(m \cdot \mathsf{len}(P))$ to $O(m)$ where m is the length of client inputs. In particular, in some rare cases — like $(n, 1)$-CPIR — there exist specific protocols like [GR05] that have better communication than $\mathsf{PrivateBP}$ with Lipmaa's $(2, 1)$-CPIR protocol. $(n, 1)$-CPIR protocol based on $\mathsf{PrivateBP}$ with a more efficient underlying $(2, 1)$-CPIR protocol would have asymptotically the same communication as the Gentry-Ramzan protocol.

(2). The class of functions that can be cryptocomputed by using $\mathsf{PrivateBP}$ in polynomial time (by client in his input length and by server in her input length) seems to be never studied before. What exactly is this class? Intuitively, only sets $\mathcal{F}$ where the maximal branching program length is superpolynomial in m, or the maximal branching program size is superpolynomial in $\log |\mathcal{F}|$ fall outside of this class. Both cases seem to be highly artificial. Moreover, it would be interesting to know when exactly is $\mathsf{PrivateBP}$ *computationally* more efficient than Yao's protocol.

(3). One can probably construct even more non-trivial examples of usefulness.

Acknowledgments. The author was supported by Estonian Science Foundation, grant #6848, European Union through the European Regional Development Fund and the 6th Framework Programme project AE-OLUS (FP6-IST-15964). We are grateful for Yuval Ishai and Eyal Kushilevitz for discussions.

References

AIR01. William Aiello, Yuval Ishai, and Omer Reingold. Priced Oblivious Transfer: How to Sell Digital Goods. In Birgit Pfitzmann, editor, *Advances in Cryptology — EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 119–135, Innsbruck, Austria, May 6–10, 2001. Springer-Verlag. 2, 3

AMG07. Carlos Aguilar-Melchor and Philippe Gaborit. A Lattice-Based Computationally-Efficient Private Information Retrieval Protocol. In *Western European Workshop on Research in Cryptology (WEWORC 07)*, pages 50–54, Bochum, Germany, June 2007. http://eprint.iacr.org/2007/446. 4.1

BCPT07. Julien Bringer, Hervé Chabanne, David Pointcheval, and Qiang Tang. Extended Private Information Retrieval and Its Application in Biometrics Authentications. In Feng Bao, San Ling, Tatsuaki Okamoto, Huaxiong Wang, and Chaoping Xing, editors, *Cryptology and Network Security, 6th International Conference, CANS 2007*, volume 4856 of *Lecture Notes in Computer Science*, pages 175–193, Singapore, December 8–10, 2007. Springer-Verlag. 4.3

BGN05. Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF Formulas on Ciphertexts. In Kilian [Kil05], pages 325–341. 1

BHR95. Yuri Breitbart, Harry B. Hunt III, and Daniel J. Rosenkrantz. On The Size of Binary Decision Diagrams Representing Boolean Functions. *Theoretical Computer Science*, 145(1&2):45–69, 1995. 1, 2, B

BIM00. Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the Servers Computation in Private Information Retrieval: PIR with Preprocessing. In Mihir Bellare, editor, *Advances in Cryptology — CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 55–73, Santa Barbara, USA, August 20–24 2000. Springer-Verlag. 4.1

BKOS07. Dan Boneh, Eyal Kushilevitz, Rafail Ostrovsky, and William E. Skeith III. Public Key Encryption That Allows PIR Queries. In Alfred Menezes, editor, *Advances in Cryptology — CRYPTO 2007, 27th Annual International Cryptology Conference*, volume 4622 of *Lecture Notes in Computer Science*, pages 50–67, Santa Barbara, USA, August 19–23, 2007. Springer-Verlag. 1, 4.2, 4.2

BPRS90. László Babai, Pavel Pudlák, Vojtech Rödl, and Endre Szemerédi. Lower Bounds to the Complexity of Symmetric Boolean Functions. *Theoretical Computer Science*, 74(3):313–323, 1990. 4.3

CGKS95. Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private Information Retrieval. In *36th Annual Symposium on Foundations of Computer Science*, pages 41–50, Milwaukee, Wisconsin, October 23–25 1995. IEEE. 1

CH08. Łukasz Chmielewski and Jaap-Henk Hoepman. Fuzzy Private Matching. In *The Third International Conference on Availability, Reliability and Security, ARES 2008*, pages ?–?, Barcelona, Spain, March 4–7, 2008. IEEE Computer Society Press. 1, 4.3

CIK$^+$01. Ran Canetti, Yuval Ishai, Ravi Kumar, Michael K. Reiter, Ronitt Rubinfeld, and Rebecca N. Wright. Selective Private Function Evaluation with Applications to Private Statistics. In *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing*, pages 293–304, Newport, Rhode Island, USA, August26–29, 2001. ACM Press. 1, 4.3, 4.3

Cob66. Alan Cobham. The Recognition Problem for the Set of Perfect Squares. In *7th Annual Symposium on Foundations of Computer Science*, pages 78–87, Berkeley, California, October 23–25, 1966. IEEE Computer Society. 2

CS07. Boris Carbunar and Radu Sion. On the Computational Practicality of Private Information Retrieval. In *The 14th Annual Network & Distributed System Security Symposium, NDSS 2007*, pages ?–?, San Diego, California, USA, February 27–March 2, 2007. 1, 4.1

DJ01. Ivan Damgård and Mads Jurik. A Generalisation, A Simplification And Some Applications of Paillier's Probabilistic Public-Key System. In Kwangjo Kim, editor, *Public Key Cryptography 2001*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136, Cheju Island, Korea, February 13–15, 2001. Springer-Verlag. 1, 2, 2

DJ03. Ivan Damgård and Mads Jurik. A Length-Flexible Threshold Cryptosystem with Applications. In Rei Safavi-Naini, editor, *The 8th Australasian Conference on Information Security and Privacy*, volume 2727 of *Lecture Notes in Computer Science*, pages 350–364, Wollongong, Australia, July 9-11, 2003. Springer-Verlag. 2

FIPR05. Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword Search And Oblivious Pseudorandom Functions. In Kilian [Kil05], pages 303–324. 2, 3

GR05. Craig Gentry and Zulfikar Ramzan. Single-Database Private Information Retrieval with Constant Communication Rate. In Luis Caires, Guiseppe F. Italiano, Luis Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *The 32nd International Colloquium on Automata, Languages and Programming, ICALP 2005*, volume 3580 of *Lecture Notes in Computer Science*, pages 803–815, Lisboa, Portugal, 2005. Springer-Verlag. 1, 4.1, 4.3, 4.3, 5

GY07. William Gasarch and Arkady Yerukhimovich. Computationally Inexpensive cPIR. Work in progress, available at `http://www.cs.umd.edu/~arkady/`, 2007. 4.1, 4.1

IP07. Yuval Ishai and Anat Paskin. Evaluating Branching Programs on Encrypted Data. In Salil Vadhan, editor, *The Fourth Theory of Cryptography Conference, TCC 2007*, volume 4392 of *Lecture Notes in Computer Science*, pages 575–594, Amsterdam, The Netherlands, February 21–24, 2007. Springer Verlag. 1, 1, 2, 3, 3

Kal05. Yael Tauman Kalai. Smooth Projective Hashing and Two-Message Oblivious Transfer. In Ronald Cramer, editor, *Advances in Cryptology — EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 78–95, Aarhus, Denmark, May 22–26, 2005. Springer-Verlag. 3

Kil05. Joe Kilian, editor. *The Second Theory of Cryptography Conference, TCC 2005*, volume 3378 of *Lecture Notes in Computer Science*, Cambridge, MA, USA, February 10–12, 2005. Springer Verlag. 5

KO97. Eyal Kushilevitz and Rafail Ostrovsky. Replication is Not Needed: Single Database, Computationally-Private Information Retrieval. In *38th Annual Symposium on Foundations of Computer Science*, pages 364–373, Miami Beach, Florida, October 20–22, 1997. IEEE Computer Society. 1, 3, 3, 4.1, A

KS05. Lea Kissner and Dawn Song. Privacy-Preserving Set Operations. In Victor Shoup, editor, *Advances in Cryptology — CRYPTO 2005, 25th Annual International Cryptology Conference*, volume 3621 of *Lecture Notes in Computer Science*, pages 241–257, Santa Barbara, USA, August 14–18, 2005. Springer-Verlag. 3

Lip05. Helger Lipmaa. An Oblivious Transfer Protocol with Log-Squared Communication. In Jianying Zhou and Javier Lopez, editors, *The 8th Information Security Conference (ISC'05)*, volume 3650 of *Lecture Notes in Computer Science*, pages 314–328, Singapore, September 20–23, 2005. Springer-Verlag. 1, 1, 2, 2, 2, 3, 3, 3, 4.1, A, B

LL07. Sven Laur and Helger Lipmaa. A New Protocol for Conditional Disclosure of Secrets And Its Applications. In Jonathan Katz and Moti Yung, editors, *5th International Conference on Applied Cryptography and Network Security – ACNS'07*, volume 4521 of *Lecture Notes in Computer Science*, pages 207–225, Zhuhai, China, June 5–8, 2007. Springer-Verlag. 3

NN01. Moni Naor and Kobbi Nissim. Communication Preserving Protocols for Secure Function Evaluation. In *Proceedings of the Thirty-Third Annual ACM Symposium on the Theory of Computing*, pages 590–599, Heraklion, Crete, Greece, July 6–8 2001. ACM Press. 3, 4.1

NP99. Moni Naor and Benny Pinkas. Oblivious Transfer and Polynomial Evaluation. In *Proceedings of the Thirty-First Annual ACM Symposium on the Theory of Computing*, pages 245–254, Atlanta, Georgia, USA, May 1–4, 1999. ACM Press. 2, 3, 4.1

OS97. Rafail Ostrovsky and Victor Shoup. Private Information Storage. In *Proceedings of the Twenty-Nineth Annual ACM Symposium on the Theory of Computing*, pages 294–303, 1997. 4.2

OS08. Rafail Ostrovsky and William E. Skeith III. Communication Complexity in Algebraic Two-Party Protocols. In David Wagner, editor, *Advances in Cryptology — CRYPTO 2008, 28th Annual International Cryptology Conference*, volume 5157 of *Lecture Notes in Computer Science*, pages 379–396, Santa Barbara, USA, August 17–21, 2008. Springer-Verlag. 1

Pai99. Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238, Prague, Czech Republic, May 2–6, 1999. Springer-Verlag. 2, 2, 1

Sha49. Claude E. Shannon. The Synthesis of Two-Terminal Switching Circuits. *The Bell Systems Technical Journal*, 28:59–98, 1949. 1, 2, 4.1, 4.1, B

ST97. Rakesh Kumar Sinha and Jayram S. Thathachar. Efficient Oblivious Branching Programs for Threshold And Mod Functions. *Journal of Computer and System Sciences*, 55(3):373–384, 1997. 1, 4.3

Ste98. Julien P. Stern. A New And Efficient All Or Nothing Disclosure of Secrets Protocol. In Kazuo Ohta and Dingyi Pei, editors, *Advances on Cryptology — ASIACRYPT '98*, volume 1514 of *Lecture Notes in Computer Science*, pages 357–371, Beijing, China, October 18–22, 1998. Springer-Verlag. 1, 3, 3, A

SYY99. Tomas Sander, Adam Young, and Moti Yung. Non-Interactive CryptoComputing For NC^1. In *40th Annual Symposium on Foundations of Computer Science*, pages 554–567, New York, NY, USA, 17–18 October 1999. IEEE Computer Society. 3

Weg00. Ingo Wegener. *Branching Programs and Binary Decision Diagrams: Theory and Applications*. Monographs on Discrete Mathematics and Applications. Society for Industrial Mathematics, 2000. 1, 2, 3, 4.1, 4.1

Yao82. Andrew Chi-Chih Yao. Protocols for Secure Computations (Extended Abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, Chicago, Illinois, USA, 3–5 November 1982. IEEE Computer Society Press. 1, 3

YYWP08. Jin Yuan, Qingsong Ye, Huaxiong Wang, and Josef Pieprzyk. Secure Computation of the Vector Dominance Problem. In Liqun Chen, Yi Mu, and Willy Susilo, editors, *Information Security Practice and Experience, 4th International Conference, ISPEC 2008*, volume 4991 of *Lecture Notes in Computer Science*, pages 319–333, Sydney, Australia, April 21–23, 2008. Springer-Verlag. 3

A Kushilevitz-Ostrovsky Reduction And Lipmaa's $(n, 1)$-CPIR Protocol

In the first step of the Kushilevitz-Ostrovsky recursion, server's original database of size n is divided into n/n' pieces of n' elements, for some $n' \ll n$. Next, a basic two-message $(n', 1)$-CPIR protocol $\Gamma' = (\mathsf{Q}', \mathsf{R}', \mathsf{A}')$ is applied to every piece. The second messages $\mathsf{R}'(\ell, \cdot, \cdot)$ of Γ' are, instead of being sent back to the client, stored at server as an intermediate database of smaller size n/n' but longer string-length $|\mathsf{R}'(\ell, \cdot, \cdot)|$. Next, the same step is applied recursively, with smaller and smaller intermediate databases of size $n/(n')^j$ of longer and longer bitlength being stored. The server only returns the answer of the final $(n', 1)$-CPIR when her database has been reduced to contain n' (long) elements.

Kushilevitz and Ostrovsky used a relatively inefficient underlying $(n', 1)$-CPIR protocol Γ' that is based on the difficulty of the quadratic residuosity problem. A more efficient $(n', 1)$-CPIR protocol, based on an arbitrary CPA-secure additively homomorphic cryptosystem was proposed by Stern [Ste98]. Finally, Lipmaa [Lip05] proposed his $(2, 1)$-CPIR protocol that was described earlier. The crucial feature of Lipmaa's $(2, 1)$-CPIR protocol is that there $|\mathsf{R}(\ell, f, \mathsf{Q})|$ grows only additively in ℓ, which makes it possible to achieve log-squared communication in the final $(n, 1)$-CPIR protocol, by applying $\log n$ recursive layers of the Kushilevitz-Ostrovsky basic recursion. In comparison, the authors of [KO97,Ste98] constructed a basic $(n', 1)$-CPIR protocol with multiplicative length expansion, which resulted in sublinear but still superpoly-logarithmic communication for the final $(n, 1)$-CPIR protocol.

Lipmaa's $(n, 1)$-CPIR protocol [Lip05] is an instantiation of the Kushilevitz-Ostrovsky recursion with Lipmaa's $(2, 1)$-CPIR protocol underlying it. For the sake of completeness, we will next give its full description. Let $t := \lceil \log n \rceil$. Let $(\mathsf{Q}', \mathsf{R}', \mathsf{A}')$ be Lipmaa's $(2, 1)$-CPIR protocol as described earlier. Lipmaa's $(n, 1)$-CPIR protocol $(\mathsf{Q}, \mathsf{R}, \mathsf{A})$ for ℓ-bit strings is depicted by Fig. 2. Let client's index be $x = \sum_{j=0}^{t-1} x_j 2^j$ for $x_j \in \{0, 1\}$. Server's input is a database $f = (f_0, \dots, f_{n-1})$. We also denote f_x by $f_{x_{t-1},\dots,x_0}$. First,

$$\mathsf{Q}(\ell, x) := \{\mathsf{Q}'(\ell + (t - 1 - j)k, x_j)\} = \{\mathsf{E}_{\mathsf{pk}}^{s+t-1-j}(x_j)\}$$

for $j \in \{0, \dots, t - 1\}$. Now, given $\mathsf{Q}(\ell, x)$, server cryptocomputes a complete ordered binary decision tree, where at the ith level, the server "branches" on the (encrypted) value of x_i. The leafs are labeled by f_i, where $f_{i_{t-1},\dots,i_0}$ is at the leaf that corresponds to the branchings made according to the tests $[x_j \stackrel{?}{=} i_j]$ being true. At every node on the jth level, the server computes $\mathsf{R}'(\ell + (t - 1 - j)k, (b_0, b_1), \mathsf{Q}'_j)$ where $\mathsf{Q}'_j := \mathsf{Q}'(\ell + (t - 1 - j)k, x_j)$. That is, given Q'_j and an input pair (b_0, b_1) to this node, the server computes the value $\mathsf{R}'(\ell + (t - 1 - j)k, (b_0, b_1), \mathsf{Q}_j) = (\mathsf{E}_{\mathsf{pk}}^{s+t-1-j}(1)/\mathsf{Q}_j)^{b_0} \cdot \mathsf{Q}_j^{b_1} = \mathsf{E}_{\mathsf{pk}}^{s+t-1-j}((1 - x_j) \cdot b_0 + x_j \cdot b_1) = \mathsf{E}_{\mathsf{pk}}^{s+t-1-j}(b_{x_j})$. This value is then used as an input at the level $j - 1$. Thus, at the end of Lipmaa's $(n, 1)$-CPIR protocol, the server obtains the value

$$\begin{aligned}\mathsf{R}(\ell, f, \mathsf{Q}(\ell, x)) \leftarrow &\mathsf{R}'(\ell + (t - 1)k, (\mathsf{R}'(\ell + (t - 2)k, \dots, \mathsf{Q}'_{t-1}), \mathsf{R}'(\ell + (t - 2)k, \dots, \mathsf{Q}'_{t-1})), \mathsf{Q}'_t) \\ = &\mathsf{E}_{\mathsf{pk}}^{s+t-1}(\mathsf{E}_{\mathsf{pk}}^{s+t-2}(\dots \mathsf{E}_{\mathsf{pk}}^{s}(f_{x_{t-1},\dots,x_0}) \dots)) \enspace ,\end{aligned}$$

and sends it to the client. The client multiple-decrypts server's message and thus obtains f_x.

B Efficiency of Shannon's Upperbound

This example is based on an OBDD that satisfies Shannon's upperbound $(3 + o(1))2^m/m$ on the size of branching programs from [Sha49], since the more precise upperbound of [BHR95] seems only to apply for $m \geq 10$. See Fig. 3 for concrete case $m = 6$. Let $f : \{0, 1\}^m \to \{0, 1\}$ be a Boolean function. The

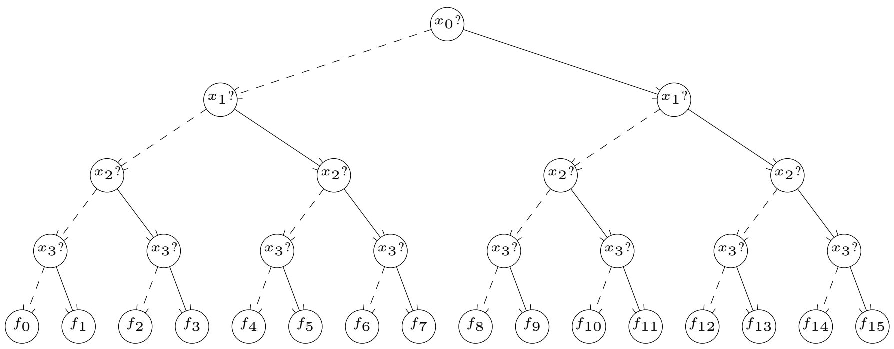
x_0?
x_1?
x_2?
x_3?
f_0 f_1 f_2 f_3 f_4 f_5 f_6 f_7 f_8 f_9 f_{10} f_{11} f_{12} f_{13} f_{14} f_{15}

Figure2. Lipmaa's CPIR is based on a complete ordered binary decision tree, for $n = 2^m = 16$. In all figures of this paper, dotted lines correspond to the branch 0 and solid lines to the branch 1

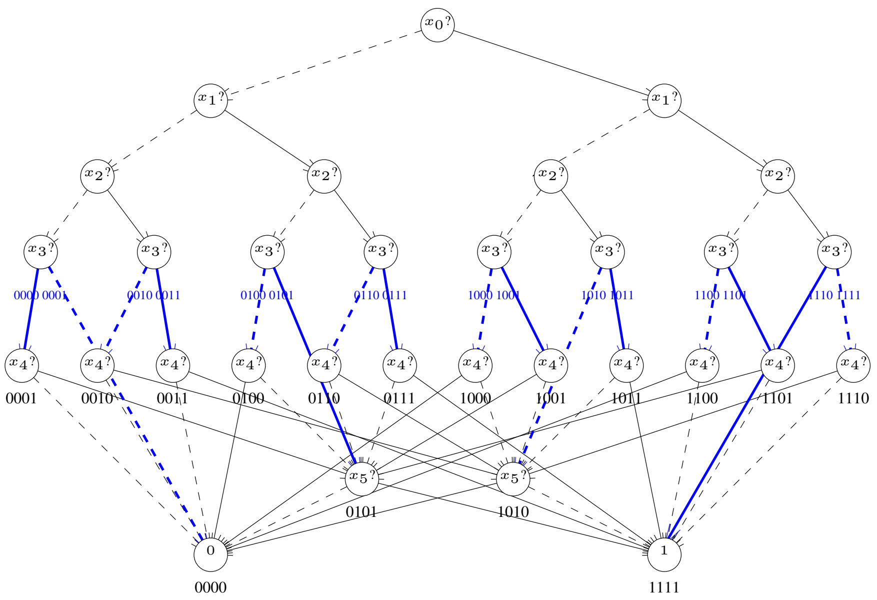

Figure3. Communication-efficient CPIR based Shannon's upper bound for $n = 2^m = 64$. Only blue values and edges depend on the concrete database, which is equal to a sequence of binary presentations of all 4-bit integers. Everything else depends just on the value of m

idea behind Shannon's construction is to construct an ordered branching program, such that: the branching program starts out as a depth d, where $d = m - \lfloor \log(m + 1 - \log m) \rfloor$, ordered binary decision tree where one branches on variables $x_0, \dots, x_{d-1}$. This results in $2^d - 1$ nodes. The branching program has $2^{2^{m-d}}$ more nodes that correspond to all subfunctions of f on last $m - d$ variables. These extra nodes are layered in $m - d$ more levels. A 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 a 2^{m-d}-bit string corresponding to 2^{m-d} values in the truth table of f. There is an 0-edge from an extra node labeled by $v_1 \dots v_{2^{m-d}}$ to an extra node $v'_1 \dots v'_{2^{m-d}}$ exactly if $v'_j = v'_{2^{m-d-1}} + j = v_j$ for $j \in \{1, 2^{m-d-1}\}$. There is an 1-edge from an extra node labeled by $v_1 \dots v_{2^{m-d}}$ to an extra node $v'_1 \dots v'_{2^{m-d}}$ exactly if $v'_j = v'_{2^{m-d-1}+j} = v_{2^{m-d-1}+j}$ for $j \in \{1, 2^{m-d-1}\}$.

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} values in the truth table of f. The 0-edge from node level d edge $v_1, \dots, v_{2^{m-d+1}}$ goes to an extra node labeled by $v_1, \dots, v_{2^{m-d}}$. The 1-edge from node level d edge $v_1, \dots, v_{2^{m-d+1}}$ goes to an extra node labeled by $v_{2^{m-d}+1}, \dots, v_{2^{m-d+1}}$. This means that only the location of $2^d \approx 2^m/(m + 1 - \log m) = (1 + o(1))2^m/m$ edges depends on the database. Thus even in the offline phase, even when the database is completely changed, one has to change $O(2^d) = O(2^m/m)$ edges, this can be compared to the 2^m work that is necessary to update the database itself. In the case the database is updated in only one element, only the location of one edge is changed.

In the concrete case $m = 6, d = m - \lfloor \log(m+1-\log m) \rfloor = 4$. Complete ordered decision tree (that is, CPIR from [Lip05]) has $2^{m+1} - 1 = 127$ nodes. The branching program based on Shannon's construction has $2^d - 1 + 2^{2^{m-d}} = 15 + 16 = 31$ nodes.