Private Branching Programs: On Communication-Efficient Cryptocomputing*

Helger Lipmaa

University College London, UK

Abstract We polish a recent cryptocomputing method that makes it possible to cryptocompute every language in **L/poly**. We give several nontrivial applications, including: (a) A CPIR protocol with log-squared communication and sublinear server-computation by giving a secure function evaluation protocol for Boolean functions with similar performance, (b) A protocol that makes it possible to compute (say) how similar is client's input to an element in server's database, without revealing any information to the server, (c) A protocol for private database updating with low amortized complexity.
Keywords. Binary decision diagrams, branching programs, computationally-private information retrieval, cryptocomputing, private database updating.

1 Introduction

A branching program [Weg00] is a fanout-2 directed acyclic graph where the internal nodes are labeled by variables from some variable set $\{\iota_1, \ldots, \iota_n\}$, 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 m sources computes some function $f : \{0,1\}^n \to \{0,1\}^{m\ell}$; $m = \ell = 1$ by default. It is well known that the class of languages that can be computed by polynomial-size branching programs is equal to the class **L/poly** of languages that can be computed by non-uniform Turing machines in logarithmic space [Cob66,Weg00].

In a (single-database) $(N, 1)$-computationally-private information retrieval (CPIR) protocol [CGKS95], the client has a query $\iota \in \{0, \ldots, N-1\}$ and the server has a database $D = (D_0, \ldots, D_{N-1})$ of ℓ-bit strings for some ℓ. At the end of the protocol, client obtains D_ι while the server remains completely clueless about the value of ι. One can privately compute any language L in the class **L/poly** by first fixing the input length n and then designing an efficient branching program for this length. After that, the local selector/branch operation at every node of the branching program is implemented by using a suitable two-message $(2, 1)$-CPIR protocol. More precisely, the client sends to the server the first message Q_i of the $(2, 1)$-CPIR protocol corresponding to client's every input variable ι_i. At every node, the server sets a secondary label of the incoming edge of this node to be equal to the second message of the $(2, 1)$-CPIR protocol that uses Q_i and the database that consists of two labels of the outcoing edges of the same node. (The secondary label of the incoming edge of a sink is just equal to the label of this sink). The secondary label of the incoming edge of the source is then returned to the client who recursively applies the local decoding procedure to this message to obtain the value of the branching program.

This *PBP* (private branching programs) protocol was first proposed in [IP07]. (The $(N, 1)$-CPIR protocols by [KO97,Ste98,Lip05] use exactly the same idea to implement a private version of a complete m-ary (multi-terminal) ordered decision tree though also this was probably first explicitly stated in [IP07].) Clearly, the PBP protocol is client-private because the server only sees first messages of n different $(2, 1)$-CPIR protocols. See Sect. 3 for a precise description of the PBP protocol.

Currently, Lipmaa's $(2, 1)$-CPIR protocol [Lip05] is the most efficient known $(2, 1)$-CPIR protocol for the purpose of the PBP protocol. If Lipmaa's protocol is used, the communication of PBP is linear in the size of *client's* input and in the length of the branching program, and assuming the RAM model, server's online computation is linear in the size of the branching program. Importantly, the PBP protocol only reveals the length (that is, the depth)—but not the shape or even the size—of the branching program to the client. This makes it possible to use the PBP to solve problems where client's input can be an arbitrary n-bit string and server's input is a function $f : \{0,1\}^n \to \{0,1\}^{m\ell}$ from some set $\mathcal{F}$ such that all functions from $\mathcal{F}$ can be computed by branching programs of polynomial size and (possibly) small length.

* Third public draft, May 15, 2008. This draft has been public solely for introducing the ideas to the public. Correlation between this draft and final publications may be very small.

Our Contributions. While the PBP protocol was proposed in [IP07], the autors did not propose many concrete applications. We aim to popularize the PBP protocol by solving a few well-known problems in cryptographic protocol design that are interesting in their own right. Note that with the next we kill two birds with one stone. First, we solve long-standing open problems in their own areas. Second, we show that the PBP protocol is useful also in practice: namely, almost direct applications of the PBP protocol—combined with results from the theory of branching programs—result in protocols that are more 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 efficient than specialized protocols for the same task. (A comparison with Yao's protocol is also given in Sect. 3.)

Computation-Efficient CPIR. It is well-known that computation-efficiency is the main bottleneck in deploying $(N, 1)$-CPIR in practice; this has motivated quite some attention on this aspect of the CPIR protocols, see e.g. [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 negative 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. This 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]. Moreover, one can implement secure function evaluation of any $f : \{0,1\}^n \to \{0,1\}$ with communication $O(n^2)$ and computation $O(2^n/n)$; the latter upperbound is also tight for all but an expontentially small fraction of Boolean functions [Weg00]. Thus, we also have a worst-case lowerbound for the online computation of CPIR for all but an exponentially small fraction of databases. The offline computational complexity is also $O(N/\log N)$ while the communication is log-squared in N. We achieve this by writing down a branching program for the function f_D, $f_D(\iota) := D_i$ and then using the PBP protocol to secure it. One can transform the CPIR protocol into an $(N, 1)$-oblivious transfer protocol, where also server's privacy is protected, by say using a computation-efficient transformation of [NP99], or a communication-efficient transformation of [NN01].

Extended CPIR. Second, we study extended $(N, 1)$-CPIR, a primitive recently defined in [BCPT07]. In extended CPIR, the client submits a private index ι and a private secret j to the server, who replies with the value $f(j, D_i)$ of a predetermined function f on j and server's database ιth element. While [BCPT07] only proposed extended CPIR protocols for a few simple functions, we show that by using the PBP protocol one can implement a large class of functionalities in polynomial-time and log-squared communication. Our results on extended CPIR can for example be applied in the next setting in biometric authentication that is known as fuzzy private matching [CH08]. The client of the extended CPIR protocol collects a fingerprint j of some person, together with her claim that she is the ιth 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 ιth 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 ι. 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 $\Theta(N \cdot \ell \log^3 \ell/\log\log\ell\log\log\log\ell)$.

Private database updating. Third, we study private database updating where on client's private index ι and a private secret j, the server updates the ιth element of his database to j. 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 Crypto 2007 [BKOS07]. Their solution was based on bilinear maps. We propose another non-trivial solution that, for u updates, achieves *amortized* communication complexity $\Theta(u^2 \log^2 N)$; this improves upon the protocol of [BKOS07] for $u = o(\sqrt[4]{N}/\log^4 N)$. Importantly, this protocol does not use bilinear pairings.

One can combine ideas of the two last protocols to let the server to store a value of some function of the previous value of the database element and of a secret input of the client. To the best of our knowledge, this problem has not been tackled in the literature at all.

Other applications. Because under many well-known assumptions there exist say primitives for computing public-key encryption, signing, collision-resistant hash functions and secret-key encryption in the complexity class $\mathbf{NC^1} \subseteq$

L/poly, one can communication-efficiently implement such primitives on encrypted data. It is however not clear whether one is interested in communication-efficiency in such applications and thus we do not elaborate on this. We mention a few other applications. For example, by using the PBP one can solve both Yao's millionaire's problem [Yao82] and the secure vector dominance problem for n-bit vectors with computation $\Theta(n)$ and communication $(n+1)(n+2)k = \Theta(n^2)k$.

Many existing cryptographic protocols are based on the use homomorphic encryption and because of that, are limited to cryptocomputing affine functions. A more recent trend is to use bilinear-map based cryptosystems [BGN05] because they make it possible to cryptocompute quadratic functions. Because Lipmaa's $(2,1)$-CPIR protocol is based on a length-flexible homomorphic cryptosystem, the PBP protocol also demonstrates that by such a cryptosystem, one can cryptocompute a much larger class of functions. In particular, the PBP protocol uses only one application of such cryptosystems (an efficient $(2,1)$-CPIR protocol); an interesting open question is to find out the precise computational power of length-flexible cryptosystems.

2 Preliminaries, Related Work, Cryptographic Tools

Notation. Within the paper, n denotes the length of client's input ι. In CPIR-like applications, N denotes the database size. All logarithms have base 2. Throughout the paper, the client is going to be a he and the server is going to be a she.

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 $\{\iota_1, \ldots, \iota_n\}$, 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 m sources computes some function $f : \{0,1\}^n \to \{0,1\}^\ell$; $m = \ell = 1$ 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. For a branching program P let $\mathsf{len}(P)$ be its length (that is, the length of its longest path) and $\mathsf{size}(P)$ be its size. Denote by $\mathsf{BP}(f)/\mathsf{OBDD}(f)$ the minimal size of any branching program/OBDD computing f. Clearly $\mathsf{BP}(f) \le \mathsf{OBDD}(f)$. It is known that any Boolean function $f : \{0,1\}^n \to \{0,1\}$ has $\mathsf{OBDD}(f) \le (3+o(1))2^n/n$ [Sha49]; for $n \ge 16$ this can be improved to $\mathsf{OBDD}(f) \le (2+o(1))2^n/n$ [BHR95] though in practice even then Shannon's construction is as efficient. For $n \ge 25$, [BHR95] proved a precise upperbound $\mathsf{BP}(f) \le (1+o(1))2^n/n$. The latter upperbound is also tight because for all but an expontentially small fraction of $2^{-2^{n/2}}$ of Boolean functions the size of the best branching program is $(1-n^{-1/2})2^n/n = (1-o(1))2^n/n$ [Weg00, Thm. 2.2.2]. A language (or a Boolean function) has a polynomial-size branching program if and only if it belongs to the complexity class **L/poly** [Cob66], that is, if it can be decided by a nonuniform log-space Turing machine. See [Weg00] for an extensive coverage of branching programs.

Public-Key Cryptosystems. Let $\mathsf{P} = (\mathsf{G}, \mathsf{E}, \mathsf{D})$ be a length-flexible additively-homomorphic public-key cryptosystem [DJ01], where for every integer $s > 0$, $\mathsf{E}^s_{\mathsf{pk}}(\cdot)$ maps a plaintext from set $\mathcal{M}_s$ to a ciphertext in set $\mathcal{M}_{s+1}$. In the case of [DJ01], $\mathcal{M}_s = \mathbb{Z}_N^s$ for a large prime N, and thus the plaintext length is sk while the ciphertext length is $(s+1)k$ bits. (In some other length-flexible cryptosystems, the resulting ciphertext is longer, for example, $(s+2)k$ bits in [DJ03].) Recall that in such a 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 $\mathsf{E}^{s'}_{\mathsf{pk}}(M)$, pk, s' and s for $s' \ge s$, returns $\mathsf{E}^{s'}_{\mathsf{pk}}(M)$. A compress function exists for the cryptosystem from [DJ01] as shown in [Lip05]: it just reduces $\mathsf{E}^{s'}_{\mathsf{pk}}(M)$ modulo $|\mathcal{M}_{s+1}|$.

In the CPA 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 C, 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 length parameters s_j. This so-called LFCPA-security [Lip05] of the cryptosystem is crucial for the efficient PBP protocol as defined in the next section. The length-flexible additively-homomorphic public-key cryptosystem by Damgård and Jurik from [DJ01] is CPA-secure under the Decisional Composite Residuosity Assumption [Pai99]. It also has the compress function C.

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 $\iota \in \{0, \ldots, N-1\}$ and the server has a database $D = (D_0, \ldots, D_{N-1})$. The client obtains D_i. We say a CPIR protocol Γ is "friendly" if it satisfies the next three assumptions:

- the protocol Γ has two messages, a query $\mathsf{Q}(\ell, \iota)$ from the client and a reply $\mathsf{R}(\ell, D, \mathsf{Q})$ from the server, such that the stateful client can recover D_ι by computing a local function $\mathsf{A}(\ell, \iota, \mathsf{R}(D, \mathsf{Q}))$
- the same protocol is uniform in ℓ, that is, it can be easily modified to work on other values of ℓ.
- there exists a compress function C that maps $\mathsf{Q}(\ell', \iota)$ to $\mathsf{Q}(\ell, \iota)$ for any $\ell' \geq \ell$ and ι.

More formally, $\Gamma = (\mathsf{Q}, \mathsf{R}, \mathsf{A}, \mathsf{C})$ is a quadruple of probabilistic polynomial-time algorithms, with $\mathsf{A}(\ell, \iota, \mathsf{R}(\ell, D, \mathsf{Q}(\ell, \iota))) = D_\iota$, and $\mathsf{C}(\ell', \ell, \mathsf{Q}(\ell', \iota)) = \mathsf{Q}(\ell, \iota)$ for any $\ell' \geq \ell$ and ι. If the existence of C is not required we also write $\Gamma = (\mathsf{Q}, \mathsf{R}, \mathsf{A})$ even if C exists.

Lipmaa's "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 $\iota\{0,1\}$, server's private input is $D = (D_0, D_1)$. The client first generates a new key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{G}$. The client sends pk and

$$\mathsf{Q}(\ell, \iota) \leftarrow \mathsf{E}^s_{\mathsf{pk}}(\iota)$$

to server, who replies with

$$\mathsf{R}(\ell, D, \mathsf{Q}) \leftarrow (\mathsf{E}^s_{\mathsf{pk}}(1)/\mathsf{Q})^{D_0} \cdot \mathsf{Q}^{D_1} \ .$$

If $\iota \in \{0,1\}$ then clearly $\mathsf{R}(\ell, D, \mathsf{Q}) = \mathsf{E}^s_{\mathsf{pk}}(D_0 \cdot (1-\iota) + D_1 \cdot \iota) = \mathsf{E}^s_{\mathsf{pk}}(D_\iota)$. If P has a compress function then also Lipmaa's 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 $\mathsf{Q}(\iota) \leftarrow \mathsf{E}^{s_i}_{\mathsf{pk}}(m_i)$ encrypted by using the same public key pk. The important properties of this "friendly" $(2,1)$-CPIR protocol that make it efficient in our applications are that $|\mathsf{Q}(\ell, \cdot)| = sk + k \approx \ell + k$ and $|\mathsf{R}(\ell, \cdot, \cdot)| = sk + k \approx \ell + k$.

$(N, 1)$-CPIR by Cryptocomputing The Complete OBDT. Kushilevitz and Ostrovsky [KO97] proposed a generic protocol for recursively computing $(N, 1)$-CPIR for ℓ-bit database elements. In the first step of 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 underlying $(2, 1)$-CPIR protocol that was described earlier. The crucial feature of Lipmaa's $(2, 1)$-CPIR protocol is that there $|\mathsf{R}(\ell, D, \mathsf{Q})|$ 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 protocols from [KO97,Ste98] constructed a basic $(N', 1)$-CPIR protocol with multiplicative length expansion, which resulted in sublinear but still superpolylogarithmic communication for the final $(N, 1)$-CPIR protocol.

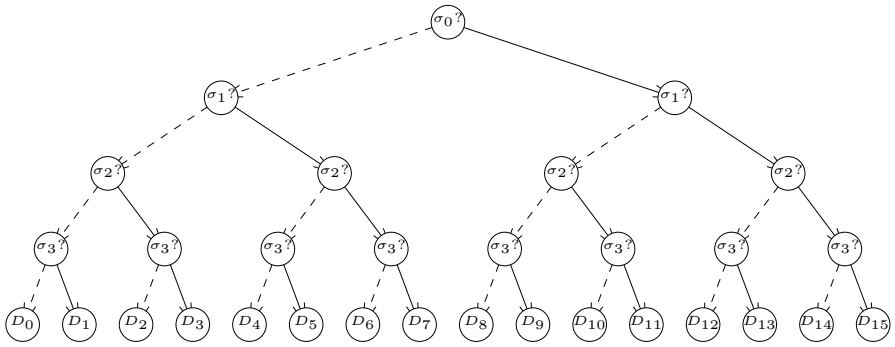


Figure1. Lipmaa's CPIR is based on a complete ordered binary decision tree, for $N = 2^n = 16$. In all figures of this paper, dotted lines correspond to the branch 0 and solid lines to the branch 1

Lipmaa's $(N, 1)$-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. 1. Let client's index be $\iota = \sum_{j=0}^{t-1} \iota_j 2^j$ for $\iota_j \in \{0, 1\}$. Server's input is a database $D = (D_0, \ldots, D_{N-1})$. We also denote D_ι by $D_{\iota_{t-1},\ldots,\iota_0}$. First,

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

for $j \in \{0, \ldots, t-1\}$. Now, given $\mathsf{Q}(\ell, \iota)$, server cryptocomputes a complete ordered binary decision tree, where at the ith level, the server branches on the value of ι_i. The leafs are labeled by D_i, where $D_{i_{t-1},\ldots,i_0}$ is at the leaf that corresponds to the branchings made according to the tests $[\iota_j =^? i_j]$ being true. At every node on the jth level, the server computes $\mathsf{R}'(\ell + (t-j)k, (b_0, b_1), \mathsf{Q}'_j)$ where $\mathsf{Q}'_j := \mathsf{Q}'(\ell + (t-j)k, \iota_j)$. That is, given Q'_j and an input pair (b_0, b_1) to this node, the server computes the value $\mathsf{R}'(\ell + (t-j)k, (b_0, b_1), \mathsf{Q}_j) = (\mathsf{E}_{\mathsf{pk}}^{s+t-j}(1)/\mathsf{Q}_j)^{b_0} \cdot \mathsf{Q}_j^{b_1} = \mathsf{E}_{\mathsf{pk}}^{s+t-j}((1-\iota_j) \cdot b_0 + \iota_j \cdot b_1) = \mathsf{E}_{\mathsf{pk}}^{s+t-j}(b_{\iota_j})$. 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

$$\begin{aligned}\mathsf{R}(\ell, D, \mathsf{Q}(\ell, \iota)) \leftarrow & \mathsf{R}'(\ell + tk, (\mathsf{R}'(\ell + (t-1)k, \ldots, \mathsf{Q}'_{t-1}), \mathsf{R}'(\ell + (t-1)k, \ldots, \mathsf{Q}'_{t-1})), \mathsf{Q}'_t) \\ = & \mathsf{E}_{\mathsf{pk}}^{s+t}(\mathsf{E}_{\mathsf{pk}}^{s+t-1}(\ldots \mathsf{E}_{\mathsf{pk}}^{s}(D_{\iota_{t-1},\ldots,\iota_0}) \ldots)) \ ,\end{aligned}$$

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

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 $\iota = (\iota_0, \ldots, \iota_{n-1})$ with $\iota_j \in \{0, 1\}$, server has an input $f : \{0, 1\}^n \to \{0, 1\}^\ell$. The client starts the protocol by sending $\mathsf{Q} := \mathsf{Q}(\ell, \iota)$ 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 $\mathsf{A}(\ell, \iota, \mathsf{R})$ 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, e.g., [FIPR05,IP07], also recommend to use this model for other cryptographic protocols. More precisely, 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 CPIR protocols that are both communication and computation-efficient.

We now give an informal definition of this notion for any cryptocomputing protocol, see say [AIR01,FIPR05,Lip05,IP07] for more details. For CPA-security (that is, privacy) of the client, a malicious nonuniform probabilistic polynomial-time server is required not to be able to distinguish between client's messages $\mathsf{Q}(\ell, \iota_0)$ and $\mathsf{Q}(\ell, \iota_1)$ corresponding to any two of client's inputs ι_0 and ι_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. 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 $\{\mathsf{Q}(\ell_j, i_{j0})\}$ and $\{\mathsf{Q}(\ell_j, i_{j1}))\}$. Lipmaa's $(2,1)$-CPIR protocol [Lip05], when based on the Damgård-Jurik cryptosystem [DJ01], is CPA-secure under the Decisional Composite Residuosity Assumption [Pai99]. Clearly, any $(N,1)$-CPIR protocol, that is based on the Kushilevitz-Ostrovsky methodology, is CPA-secure if the underlying CPIR is CPA-secure. In particular, Lipmaa's $(N,1)$-CPIR protocol [Lip05] is CPA-secure under the Decisional Composite Residuosity Assumption. A semisimulatable $\binom{N}{1}$-CPIR protocol is also known as an $(N,1)$-oblivious transfer protocol.

3 PBP: Cryptocomputing for L/poly

In this section, we describe the PBP (*private branching programs*) cryptocomputing protocol from [IP07] that generalized the cryptocomputing process, done in several previous CPIR protocols, to a branching program that computes an arbitrary functionality. Our exposition is simpler than the more general exposition of [IP07] with almost straightforward security proofs. Moreover, the concrete protocol has some small differences compared to the protocol of [IP07]. In this protocol, client has private input $\iota \in \{0,1\}^n$, server has private input $f : \{0,1\}^n \to \{0,1\}^{m\ell}$ from some known set $\mathcal{F}$ of functions, and client will receiver private output $f(\iota)$. The corresponding security definition, semisimulatability, was given in Sect. 2.

Let $\Gamma = (\mathsf{Q}', \mathsf{R}', \mathsf{A}', \mathsf{C}')$ be a "friendly" $(2,1)$-CPIR protocol (see Sect. 2). Denote $|\mathsf{Q}^{(1)}(\ell)| := |\mathsf{Q}'(\ell, \iota)|$ and $|\mathsf{Q}^{(j+1)}(\ell)| := |\mathsf{Q}'(|\mathsf{Q}^j(\ell)|, \iota)|$. Moreover, define $|\mathsf{R}^{(j)}(\ell)| := |\mathsf{R}'(|\mathsf{Q}^{(j)}(\ell)|, \iota, \mathsf{Q}')|$. We assume that those values are well-defined, that is, they do not depend on the concrete values of ι and D; because Γ has to be secure, this assumption is reasonable. Let $\mathcal{F} = \{f : \{0,1\}^n \to \{0,1\}^{m\ell}\}$ be a set of functions, where every $f \in \mathcal{F}$ can be computed by a branching program P_f. Let $\mathsf{len}(\mathcal{F}) := \max_{f\in\mathcal{F}} \mathsf{len}(P_f)$ and $\mathsf{size}(\mathcal{F}) := \max_{f\in\mathcal{F}} \mathsf{len}(P_f)$. (As previously, ℓ is the length of the sink labels in bits and m is the number of sources.)

The server executes the branching program P_f bottom-up, that is, from the sinks to the source. The input values of the nodes just above the sinks are equal to the labels of the corresponding sinks. At every node of the branching program, for which the two input values are already known but the output value is not yet known, the server uses Γ to obliviously propagate one of the two input values upwards as the input value of the node at the other end of the incoming edge. The server does this for all nodes in some (say breadth-first or depth-first) order, and then sends the output values of the 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 ι. A complete description of the PBP protocol follows:

1. **Inputs:** server knows a function $f : \{0,1\}^n \to \{0,1\}^{m\ell}$ from $\mathcal{F}$ and client knows $\iota \in \{0,1\}^n$.
2. **Offline phase:** server computes an efficient branching program P_f for f that has m sources and has ℓ-bit sink labels; the client knows $\mathsf{len}(\mathcal{F})$. Let $\ell' := |\mathsf{Q}^{(\mathsf{len}(\mathcal{F})-1)}(\ell)|$.
3. **Online phase:**
   (a) **Client does:** For $j \in \{0,\ldots,n-1\}$, set $\mathsf{Q}_j \leftarrow \mathsf{Q}'(\ell', \iota_j)$. Send $\mathsf{Q}(\ell,\iota) \leftarrow (\mathsf{Q}_0,\ldots,\mathsf{Q}_{n-1})$ 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 $\mathsf{R}_v = \perp$.
      ii. Do by following some ordering of the nodes:
         A. Let v be some node with $\mathsf{R}_v = \perp$ with kids 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 ι_i and edges from v to v_0/v_1 are labeled by $0/1$.
         C. Compute and store $\mathsf{R}_v \leftarrow \mathsf{R}(\ell^*, (\mathsf{R}_{v_0}, \mathsf{R}_{v_1}), \mathsf{C}(\ell', \ell^*, \mathsf{Q}_i))$, where $\ell^* \leftarrow \max(|\mathsf{R}_{v_0}|, |\mathsf{R}_{v_1}|)$.
         D. If v is a source then 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.

Theorem 1 (Security And Efficiency Of PBP). *Assume that $\Gamma = (\mathsf{Q}', \mathsf{R}', \mathsf{A}', \mathsf{C}')$ is a CPA-secure "friendly" $(2,1)$-CPIR protocol. Then $\mathcal{F}$ has a CPA-secure cryptocomputing protocol with communication $n \cdot |\mathsf{Q}^{(\mathsf{len}(\mathcal{F}))}(\ell)| + m \cdot |\mathsf{R}^{(\mathsf{len}(\mathcal{F}))}(\ell)|$ and online computation $\mathsf{size}(\mathcal{F})$ (in the RAM model).*

Proof. Security proof is trivial because the $(2,1)$-CPIR is CPA-secure. Client's communication is $n \cdot |\mathsf{Q}^{(\mathsf{len}(\mathcal{F}))}(\ell)|$ bits, server's communication is $m \cdot |\mathsf{R}^{(\mathsf{len}(\mathcal{F}))}(\ell)|$ bits. Server has to do some work per every node of P_f. □

Note that because of the compress function C, one does not have to submit the values $\mathsf{Q}(\ell, \iota_j)$ for several different values of ℓ. If C does not exist, the client may have to submit up to $\mathsf{len}(P)$ different encryptions of every ι_j, which can increase the communication by a factor of $\mathsf{len}(P)$.

Finally, the client is completely oblivious of the shape of the branching program, except the length of it. He just encrypts his input bits by using a length parameter that depends on the length of the branching program (and on the output length ℓ), and then receives multiple-encryptions of the outputs. This in particular means that client's inputs can be re-used in many different branching programs, and that the shape of the branching program can depend on server's inputs. In general, this means that the function f itself can be seen as Bob's secret input.

Corollary 1 (Efficient Instantiations of PBP). *Assume that $\Gamma = (\mathsf{Q}', \mathsf{R}', \mathsf{A}', \mathsf{C}')$ is a CPA-secure "friendly" $(2,1)$-CPIR protocol such that $|\mathsf{Q}'(\ell, \iota)| = (1 + o(1))\ell$ and $|\mathsf{R}'(\ell, D, \mathsf{Q}')| = \Theta(\ell)$. Let $\mathcal{F}$ be a set of functions $f : \{0,1\}^n \to \{0,1\}^{m\ell}$ such that every f can be computed by a polynomial-size branching program P_f, i.e., belongs to* $\mathbf{L/poly}$. *Then $\mathcal{F}$ has a CPA-secure cryptocomputing protocol with linear-in-$\mathsf{size}(\mathcal{F})$ communication and polynomial-in-n computation.*

Proof. Because $|\mathsf{Q}'(\ell, \iota)| = (1 + o(1))\ell$ we also have $|\mathsf{Q}^{(j)}(\ell)| = (1 + o(1) \cdot j)\ell$ for any polynomially large j, and thus $|\mathsf{R}^{(j)}(\ell)| = |\mathsf{R}'((1 + o(1) \cdot j)\ell, D, \mathsf{Q}')| = \Theta((1 + o(1) \cdot j)\ell)$. By Thm. 1, PBP has then communication $\Theta(n \cdot \ell)$, where n is client's input size. Polynomial computation follows from the fact that every language in $\mathbf{L/poly}$ can be computed by a family of polynomial-sized branching programs. □

For the sake of concreteness 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 in fact the only known protocol that satisfies all the requirements of the previous corollary. A precise result follows:

Corollary 2 (PBP with Lipmaa's $(2,1)$-CPIR). *Assume that the Decisional Composite Residuosity Assumption is true [Pai99]. Let $\mathcal{F}$ be a set of functions $f : \{0,1\}^n \to \{0,1\}^{m\ell}$, and let P_f be some m-source branching program with ℓ-bit sink labels that computes f. Then $\mathcal{F}$ has a CPA-secure cryptocomputing protocol with communication upperbounded by $(n+m)(\ell + (\mathsf{len}(\mathcal{F}) + 2)k)$, and computation $\mathsf{size}(\mathcal{F})$ (in the RAM model). In particular, if $\mathsf{len}(\mathcal{F})$ is polylogarithmic in n and $\mathsf{size}(\mathcal{F})$ is polynomial in n then also communication is polylogarithmic in n and computation is polynomial in n.*

Proof. Security proof is trivial because Lipmaa's $(2,1)$-CPIR is CPA-secure. Computation is also trivial. To calculate the communication efficiency, note that $\mathsf{Q}_j = \mathsf{Q}'(\ell', \iota_j) = \mathsf{E}_{\mathsf{pk}}^{\lceil \ell/k \rceil + \mathsf{len}(\mathcal{F})}(\iota_j)$. Thus, $|\mathsf{Q}_j| = |\mathsf{E}_{\mathsf{pk}}^{\lceil \ell/k \rceil + \mathsf{len}(\mathcal{F})}(\iota_j)| = (\lceil \ell/k \rceil + \mathsf{len}(\mathcal{F}) + 1)k \le \ell + (\mathsf{len}(\mathcal{F}) + 2)k$. Thus, client sends at most $n \cdot (\ell + (\mathsf{len}(\mathcal{F}) + 2)k)$ bits. The output of the branching program is equal to m $\mathsf{len}(\mathcal{F})$-times encryptions of sink values, where the sinks are selected by the encrypted client inputs ι_j. Server's communication consists of m $\mathsf{len}(\mathcal{F})$ times encrypted messages of length $\ell + (\mathsf{len}(\mathcal{F}) + 2)k$. □

Note that if $m = \ell = 1$ then the communication can be upperbound by $(n+1)(\mathsf{len}(P) + 2)k$.

If the branching programs are read-once (that is, at every path there is at most one branch by every variable) then $\mathsf{len}(\mathcal{F}) \le n$ and the communication is upper-bounded by $(n+m)(\ell + (n+2)k) = \Theta((n+m)(\ell + nk))$. More generally, if the branching programs are read-ρ-times then the communication is upper-bounded by $(n+m)(\ell + (\rho n + 2)k) = \Theta((n+m)(\ell + \rho nk))$. Unfortunately, it is well-known that read-ρ-times branching programs are strictly weaker than read-$(\rho+1)$-times branching programs [Tha98], and in particular it seems that there is no better than polynomial-size upperbound on $\mathsf{len}(\mathcal{F})$ in general.

Simple Example: Secure Vector Dominance Problem. Assume that the client has a vector $\iota = (\iota_0, \ldots, \iota_{n-1})$ and the server has a vector $D = (D_0, \ldots, D_{n-1})$. The client has to compute a Boolean function $f_D(\iota) = 1$ if $\iota_j \ge D_j$ for all j, and $f_D(\iota) = 0$ otherwise. This can be done as follows. Let $\mathcal{F} = \{f_D : D \in \{0,1\}^n\}$. For arbitrary ι, D, $f_D(\iota)$ can be computed by a branching program P_{f_D} that has $\mathsf{size}(P_{f_D}), \mathsf{len}(P_{f_D}) = n$. Thus assuming the use of Lipmaa's $(2,1)$-CPIR protocol, there exists a protocol for secure vector dominance that has communication $(n+1)(n+2)k = \Theta(n^2)k$ and server-computation $\Theta(n)$ (in the RAM model). 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 PBP protocol for Yao's millionaire's problem [Yao82] with exactly the same complexity.

Optimizations. *Homomorphic Add-Ons.* Because we use a homomorphic cryptosystem, the server can apply homomorphic operations to the sinks of the branching program. For example, she can let the sink label to be an encryption of 0 if and only if a certain client's input decrypts to some value held by her — this can be done by using a private equality test protocol [LL07]. Or, it can be equal to the ciphertext of an affine function of client's inputs with constants being chosen by the server. Note that this is interesting only if the branching program is allowed to have nonbinary sink values. *Randomization.* Clearly, any $f \in \mathcal{F}$ can be a probabilistic function $f : \{0,1\}^n \times \mathsf{Random} \to \{0,1\}^{m\ell}$ because the server can randomize the choice of P_f. This makes it possible to cryptocompute sets of functions in $\mathbf{RL/poly}$. *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, and on the other hand, makes security proofs more complicated. As a plus-side, however, using suitable $(N', 1)$-CPIR protocols in every inner node of the branching program makes it possible to generalize the approach from using length-flexible cryptosystems to the existence of suitable $(2, 1)$-CPIR protocols. See [IP07] for more details.

Stronger Security Guarantees. One can use any of the known methods [NP99,AIR01,Kal05,Lip05,IP07,LL07] to transfer the PBP protocol from a client-private protocol (that corresponds to using a computationally-private information retrieval) to a private protocol (that corresponds to using an oblivious transfer protocol). "Semisimulated" privacy in malicious model is a standard security notion for oblivious transfer protocols [AIR01,Lip05,LL07], 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].

Comparison to Other Cryptocomputing Protocols. The PBP protocol can be compared with some other known protocols. By using Yao's "garbled circuit" approach [Yao82], one can securely compute every function f in $\mathbf{BPP/poly}$ by using communication that is linear in the circuit complexity $C(f)$ of f. In the PBP protocol, one can cryptocompute every function f in a (probably) smaller class $\mathbf{L/poly}$ but by using communication that is linear in the branching program *length* $\mathsf{len}(f)$, which is often significantly smaller than $C(f)$. Moreover, in the PBP protocol only the length $\mathcal{F}$ is revealed, while in Yao's protocol, the shape of the circuit is revealed. 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.

The difference between classes $\mathbf{BPP/poly}$ and $\mathbf{L/poly}$ could be a largely theoretical concern: although $(\mathbf{BPP/poly}) \setminus (\mathbf{L/poly})$ seems to contain interesting functionalities like extended Euclidean algorithm, decision version of gcd and $\mathbf{P}$-complete problems like CVP and linear programming, the circuit size for such problems is usually large enough for the garbled circuit approach to become impractical.

A more valid concern is the computation cost, since in Yao's protocol, one has to execute $\Theta(C(f))$ private-key operations and only $\Theta(n)$ public-key operations, where n is again client's input size. On the other hand, in the PBP protocol, one has to execute $\Theta(\mathsf{BP}(f))$ public-key operations. Because public-key operations are in general much more costly than private-key operations, this may limit the use of the PBP protocol unless communication complexity is of paramount importance. On the other hand, there are situations were 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 PBP 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 additively homomorphic cryptosystem. However, the communication in their protocol is exponential in $\mathsf{len}(f)$ while in the PBP 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 $c \geq \log n$, where n 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 Computation-Efficient $(N, 1)$-CPIR Protocol

Computation-efficiency is currently the main bottleneck in deploying $(N, 1)$-CPIR in practice and this has motivated quite some attention on this aspect of the $(N, 1)$-CPIR protocols, see e.g. [Lip05,CS07]. This is mainly because in the all known sublinear–communication $(N, 1)$-CPIR protocols, one has to apply at least one public-key operation per database element. 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. Note that if one uses the $NN1$-CPIR protocols of [Lip05,GR05] for ℓ-bit strings with $\ell > n = \log N$, 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 next 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.

Assume that $\ell = m = 1$ and the database size is $N = 2^n$, that is, that the server's database consists of 2^n 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 $\iota \in \{0,1\}^n$ and the server has a Boolean function $f_D : \{0,1\}^n \to \{0,1\}$, such that $f_D(\iota) = D_\iota$. The client needs to retrieve $f_D(\iota)$. It is known that any Boolean function f can be computed by an OBDD P_f of size $(3 + o(1))2^n/n$ that has length n [Sha49]. *Offline* computation of such a branching program may take 2^n time, but *online* evaluation of the branching program on concrete input ι takes thus $\Theta(2^n/n)$ operations in the worst case, and often much less.

In particular, when applying the PBP protocol, one needs to do $\mathsf{size}(P_f)$ online public-key operations. Because in this case $\mathsf{size}(P_f) \leq (3+o(1))2^n/n$, the number of online public-key operations is always less than 2^n, for any possible database D. To the best of our knowledge, this is the first $(N, 1)$-CPIR with this property.

Note that the upperbound $\Theta(2^n/n)$ is also tight because for all but an expontentially small fraction of $2^{-2^{n/2}}$ of Boolean functions the size of the best branching program is $(1 - n^{-1/2})2^n/n = (1 - o(1))2^n/n$ [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. However, many real-life databases may follow to this exponentially small fraction due to the redundancy present in almost all such data.

For the sake of concreteness, we state the result only in the case we use Lipmaa's underlying $(2, 1)$-CPIR protocol.

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

Proof. Follows from Cor. 2 and Shannon's upperbound, by letting $\mathcal{F}$ to be the set of all Boolean functions $f : \{0,1\}^n \to \{0,1\}$. □

Example: Efficiency of Shannon's upperbound. This example is based on an OBDD that satisfies Shannon's upperbound $(3 + o(1))2^n/n$ on the size of branching programs from [Sha49], since the more precise upperbound of [BHR95] seems only to apply for $n \geq 10$. See Fig. 2 for concrete case $n = 6$. Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function. The idea behind Shannon's construction is to construct an ordered branching program, such that: the branching program starts out as a depth d, where $d = n - \lfloor \log(n + 1 - \log n) \rfloor$, ordered binary decision tree where one branches on variables $\iota_0, \ldots, \iota_{d-1}$. This results in $2^d - 1$ nodes. The branching program has $2^{2^{n-d}}$ more nodes that correspond to all subfunctions of f on last $n - d$ variables. These extra nodes are layered in $n - d$ more levels. A node for a subfunction that first essentially depends on the jth variable out of these $n - d$ variables (but not on earlier ones) is on level $d + j$; nodes that correspond to constant subfunctions are on level n. The extra nodes are labelled by a 2^{n-d}-bit string corresponding to 2^{n-d} values in the truth table of f. There is an 0-edge from an extra node labeled by $x_1 \ldots x_{2^{n-d}}$ to an extra node $x'_1 \ldots x'_{2^{n-d}}$ exactly if $x'_j = x'_{2^{n-d-1}+j} = x_j$ for $j \in \{1, 2^{n-d-1}\}$. There is an 1-edge from an extra node labeled by $x_1 \ldots x_{2^{n-d}}$ to an extra node $x'_1 \ldots x'_{2^{n-d}}$ exactly if $x'_j = x'_{2^{n-d-1}+j} = x_{2^{n-d-1}+j}$ for $j \in \{1, 2^{n-d-1}\}$.

The above part of the construction only depends on the value of $N = 2^n$ and not on the concrete database. The next part depends on the database: The 2^{d-1} nodes on level d are labelled by subsequent 2^{n-d+1} values in the truth table of f. The 0-edge from node level d edge $x_1, \ldots, x_{2^{n-d+1}}$ goes to an extra node labeled by $x_1, \ldots, x_{2^{n-d}}$. The 1-edge from node level d edge $x_1, \ldots, x_{2^{n-d+1}}$ goes to an extra node labeled by $x_{2^{n-d}+1}, \ldots, x_{2^{n-d+1}}$. This means that only the location of $2^d \approx 2^n/(n + 1 - \log n) = (1 + o(1))2^n/n$ edges depends on the database. Thus even in the

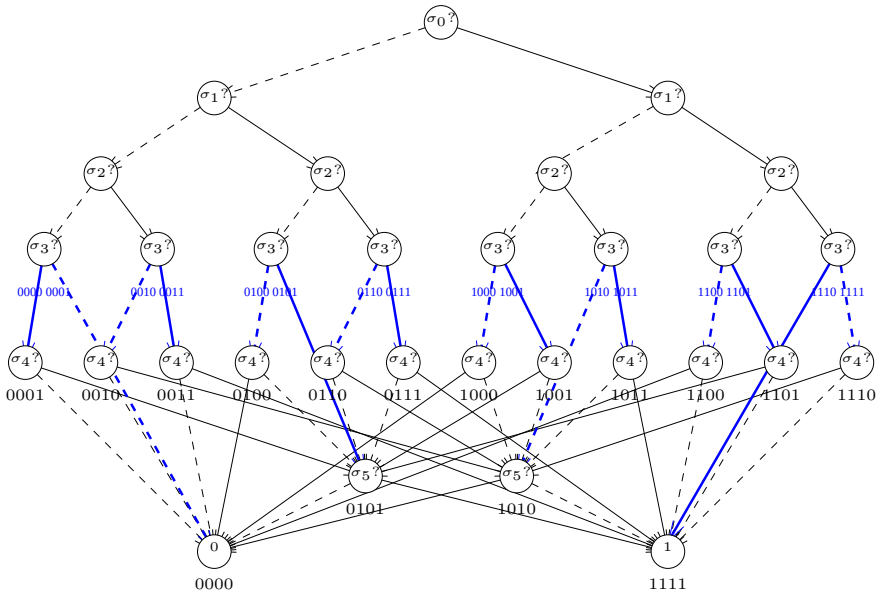


Figure2. Communication-efficient CPIR based Shannon's upper bound for $N = 2^n = 64$. Only blue values and edges depend on the concrete database, which is equal to a sequence of binary presentations of all 4-bit integers. Everything else depends just on the value of n

offline phase, even when the database is completely changed, one has to change $O(2^d) = O(2^n/n)$ edges, this can be compared to the 2^n 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 $n = 6$, $d = n - \lfloor \log(n + 1 - \log n) \rfloor = 4$. Complete OdT (that is, CPIR from [Lip05]) has $2^{n+1} - 1 = 127$ nodes. The branching program based on Shannon's construction has $2^d - 1 + 2^{2^{n-d}} = 15 + 16 = 31$ nodes.

Computation-Efficient CPIR for ℓ-Bit Strings. Let $f : \{0,1\}^n \to \{0,1\}^\ell$ for an arbitrary $\ell \geq 1$. By the upper bound of [Sha49], clearly $\mathsf{BP}(f) \leq \ell \cdot (3 + o(1))2^n/n$ by just computing ℓ branching programs in parallel. By following the proof of [Weg00], one can easily show that $\mathsf{BP}(f) \leq (3 + o(1))2^n \cdot \ell/(n + \log \ell)$. 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 though this approach seems to be not as good for large values of ℓ.

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 [NP99,AIR01,Kal05,Lip05,IP07,LL07]. In this case, we recommend the use of the transformation from [NP99] (if we are interested in computation-efficiency) or the transformation from [NN01] (if we are 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, see the original papers.

5 Extended CPIR

Preliminaries. In [BCPT07] the authors considered the problem of *extended CPIR*, where the client obtains, instead of the database element D_ι itself, some function $f(j, D_\iota)$ of client's second input and the database element. The authors proposed efficient solutions for equality (based on homomorphic encryption) and Hamming weight (based on 2-homomorphic encryption [BGN05]). Assume $J := |j|$. Moreover, for an arbitrary f there exists the next trivial protocol where the database server constructs a new database $D'_{j,\iota}$ such that $D'_{j,\iota} = f(j, D_\iota)$, and the client obtains $D'_{j,\iota}$ by doing a CPIR to this new database. Because $\log N + J$ may be quite large, the trivial protocol takes $N \cdot 2^J$ computation and $\Theta(\log N + J + |f(\cdot,\cdot)| + k)$ communication by using the Gentry-Ramzan $(N, 1)$-CPIR protocol [GR05]. The new protocol, described below, is significantly more efficient.

New Protocol. Assume that client has two inputs ι and j, and server has a database $D = (D_0, \ldots, D_{N-1})$. The client must obtain $f(j, D_\iota)$. Intuitively, we construct a protocol for extended CPIR by plugging in branching programs $P_{f,j}$ for computing $f(j, D_\iota)$, instead of just a leaf with D_ι, at the bottom of the complete OBDT for computing the $(N, 1)$-CPIR itself. The resulting protocol has length $t + \max_j \mathsf{len}(P_{f,j})$ and size $N - 1 + \sum_j \mathsf{size}(P_{f,j})$.

Optimizations. One can optimize this solution by using non-generic techniques. First, the top OBDT can be directly replaced by any secure CPIR protocol, like the protocol of Gentry and Ramzan [GR05]. Second, based on the fact that all computation uses homomorphic encryption, one can use homomorphic operations at the bottom of the branching program. For example, to compute equality $f(j, D_\iota) = [j =^? D_\iota]$, one can input as leaf the values $\star(j - D_\iota)$, where $\star$ denotes a suitable random value (a random group element if Elgamal is used, or a slight variation of it when Paillier/Damgård-Jurik is used [LL07]). Third, one is not restricted to this shape of a branching program; for many functions f we can definitely construct a more efficient branching program for the extended CPIR by further optimizing.

Example: Comparison. As an example, suppose that the client wants to establish whether $j > D_\iota$, $j = D_\iota$ or $j < D_\iota$, where all values are ℓ-bit long. One can construct a branching program for this with length ℓ and size $2\ell + 1$. Thus without optimisation, this method results in a PBP protocol with communication $\approx (\log N{+}1)((\log N{+}\ell)k{+}\ell) = \Theta(\log^2 N \cdot k + N\ell)$ and computation $\Theta(N \cdot \ell)$. If one uses the Gentry-Ramzan $\binom{N}{1}$-CPIR protocol on the top, the total communication becomes $N(\ell k + \ell) + O(\log N + \ell k + \ell + k)$.

Fuzzy private matching. As another example, suppose that the client wants to establish whether $w_h(j, D_\iota) < t$, 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 a threshold function $T_{\ell,t}$, where $T_{\ell,t}(x_1, \ldots, x_\ell) = 1$ iff at least t bits x_i are equal to 1.

It is well-known that one can construct an ordered branching program for the threshold function with length ℓ and size $\ell(\ell - t - 1) + 2 \leq \ell^2/2 - \ell/2 + 2$ [Weg00], see Fig. 3 (left). The total size of the branching program for fuzzy matching is thus $\Theta(N \times \ell^2)$, and its length is $\log N + \ell$. Thus according to Thm. 1, we have a cryptocomputing protocol for fuzzy matching with communication

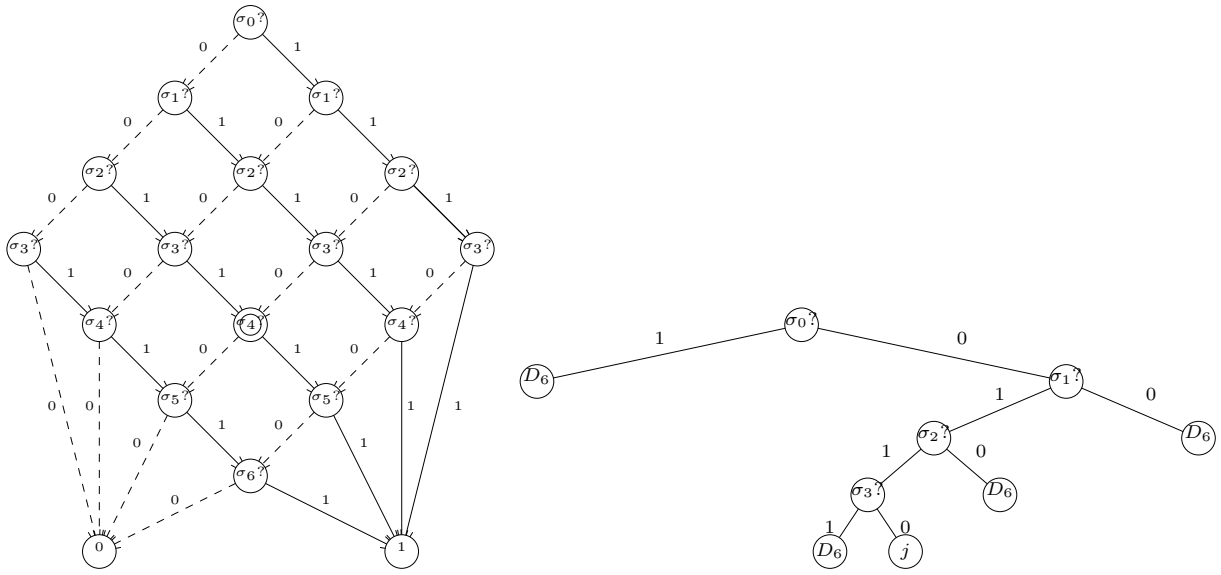
$$\leq (n+1)(\mathsf{len}P + 2)k = (\log N + \ell + 1)(\log N + \ell)k = \Theta(\log^2 N + \ell \cdot \log N + \ell^2)k$$

and computation $\Theta(N \cdot \ell^2)$. One can use the Gentry-Ramzan CPIR protocol in the upper level to decrease the communication. Moreover, better branching program 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]. (See also [ST97] for the known lower bounds on the size/length of branching program for threshold functions.) Combining their solution with the Gentry-Ramzan CPIR protocol results in a fuzzy matching protocol with communication $\Theta(\ell^2 \log^2 \ell / \log\log\ell \log\log\log\ell + \log N)k = \tilde{\Theta}(\ell^2 + \log N)k$ and computation $\Theta(N \cdot \ell \log^3 \ell / \log\log\ell \log\log\log\ell)k = \Theta(N\ell)$ which compares very favorable 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 $\Theta(\ell + \log N)$.

6 Private Database Updating

Assume that client has outsourced his database to the server. Due to the privacy requirements, the database is encrypted. In a private database updating (PDU) 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 (ι, ν), server has an encrypted database $D = (D_0, \ldots, D_{N-1}))$. The client has no output, while server obtains a new encrypted database D' such that D'_i and D_i decrypt to the same value if $i \neq \iota$, while D'_ι decrypts to ν.

It is trivial to construct a PDU protocol with $\Theta(N)$ communication by just letting the server to send the old database to client, the client to update the ιth element, re-encrypt the database and send the new database back. The next (known) approach uses an additively homomorphic cryptosystem is used. The client and the server first execute a CPIR protocol, so that the client obtains the current value of D_ι. Then client forwards to the server N ciphertexts c_j, where c_ι decrypts to $\nu - D_\iota$ and other c_j-s decrypt to 0. The server multiplies encryptions of D_j with



(a) Ordered branching program of size $\Theta(\ell^2)$ for $T_{\ell,t}$ with $\ell = 7, t = 4$

(b) A branching program that returns j if $\iota = 6$, and D_6 otherwise

Figure3. Some more examples

the new ciphertexts c_j. 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.

New Protocol. By using the PBP protocol, we can achieve amortized communication $u^2 \cdot \log^2 N$ if the number of updates is upper-bounded by u. Moreover, every update only consists of a single message from the client to the server, and the computation is also reasonable. The idea follows. Client's inputs are $(\iota_0, \ldots, \iota_{t-1}; j)$. Server's input is an encrypted database D. The server runs N branching programs P_i in parallel, where P_i returns D_i if $i \neq \iota$, and j otherwise. (See Fig. 3, right.) Let the new database be D'. Instead of returning D' to the client, the server stores D' instead of D.

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 complexity $(t^2+t) \cdot k + (t+1)(tk+k+\ell)$. Analogously, the ith update protocol has communication complexity $(t^2+t) \cdot k + (t+1)(itk+ik+\ell)$, and thus the first u protocols have total communication complexity $u(t^2+t) \cdot k + (t+1) \sum_{i=0}^{u-1} (itk + ik + \ell) = O(t^2 u^2 k)$. Thus this protocol is more communication-efficient than the protocol of [BKOS07] if $u \leq \sqrt[4]{N}/\log^4 N$. Moreover, it does *not* use pairings. In fact, 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.

7 Open Questions

Currently, Lipmaa's $(2,1)$-CPIR protocol is the most efficient underlying protocol. The PBP protocol can be made more communication-efficient if we had a "friendly" $(2,1)$-CPIR protocol where client's first message's length did not depend on ℓ, the length of database elements. One could hope to achieve this by designing 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 $O(n \cdot \mathsf{len}(P))$ to $O(n)$ where n is the length of client inputs.

Construct even more non-trivial examples of usefulness.

Acknowledgment. The author was partially supported by the Estonian Science Foundation grant 6848. 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.

[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.

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

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

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

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

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

[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.

[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.

[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.

[Tha98] Jayram S. Thathachar. On Separating the Read-k-Times Branching Program Hierarchy. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing*, pages 653–662, New York, May 23–26, 1998.

[Weg00] Ingo Wegener. *Branching Programs and Binary Decision Diagrams: Theory and Applications*. Monographs on Discrete Mathematics and Applications. Society for Industrial Mathematics, 2000.

[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.

[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.