Oblivious Decision Programs from Oblivious Transfer: Efficient Reductions

Payman Mohassel[1] and Salman Niksefat[2]

[1] University of Calgary
pmohasse@cpsc.ucalgary.ca
[2] Amirkabir University of Technology
niksefat@aut.ac.ir

Abstract. In this paper, we design efficient protocols for a number of *private database query* problems. Consider a general form of the problem where a client who holds a private input interacts with a server who holds a private decision program (e.g. a decision tree or a branching program) with the goal of evaluating his input on the decision program without learning any additional information. Many known private database queries such as Symmetric PIR, and Private Keyword Search can be formulated as special cases of this problem.

We design *computationally efficient* protocols for the above general problem, and a few of its special cases. In addition to being one-round and requiring a small amount of work by the client (in the RAM model), our protocols only require a small number of exponentiations (independent of the server's input) by both parties. Our constructions are, in essence, efficient and black-box reductions of the above problem to 1-out-of-2 oblivious transfer. We prove our protocols secure (private) against *malicious* adversaries in the standard ideal/real world simulation-based paradigm.

The majority of the existing work on the same problems focuses on optimizing communication. However, in some environments (supported by a few experimental studies), it is the computation and not the communication that may be the performance bottleneck. Our protocols are suitable alternatives for such scenarios.

1 Introduction

The *client/server paradigm* for computation and data retrieval is arguably the most common model for interaction over the internet. The majority of the services currently provided over the web are laid out in this framework wherein an often more resourceful entity (i.e. the server) provides its services to a large pool of clients. The need for the client/server model is even more justified given the widespread use of (small) mobile devices with varying computational and storage capacities.

Most client-server applications, in one way or another, deal with personal and/or sensitive data. Hence, it is not surprising that the protocols designed in this model have been the subject of extensive study by the cryptographic community. A few notable examples include private information retrieval and its extensions [15,9,13], or the more recent effort on securely outsourcing computation [10,7].

Communication vs. Computation. Consider the problem of symmetric private information retrieval (SPIR) [15,5,16]. SPIR refers to a PIR scheme with the additional security requirement that the server's database also be kept private. The majority of the research on this problem is focused on improving the communication complexity, because communication between the client and the server is often considered to be the most expensive resource. Despite achieving this goal, other barriers continue to limit realistic deployment of SPIR schemes; the most limiting of which is computation. In particular, while servers often have higher computational resources, they also need to serve a large pool of clients; consequently, even a small increase in the computation of the server for a single run of the protocol, negatively affects its overall performance. Furthermore, a number of experimental studies [24,22] conclude that, in many network setups where private database queries are likely to be deployed, it is the computation (and not the communication) that might be the performance bottleneck.[1]

Unfortunately, given the security requirements for SPIR schemes (or even PIR), it is possible to show that the server's work has to be at least linear in the size of his database (e.g. see [3]). Hence, there is no hope of achieving better asymptotic efficiency. Nevertheless, the type of operations (e.g. asymmetric vs. symmetric-key) the server performs has a significant effect on the efficiency of the resulting scheme. This is particularly important in real applications since based on existing benchmarks (e.g. http://bench.cr.yp.to) asymmetric operations (e.g. exponentiation) require *several thousand* times more cpu cycles compared to their symmetric-key counterparts. In all the constructions we are aware of for SPIR, except for one, the number of *exponentiations* the server has to perform is at least linear in the size of his database. The exception is the construction of Naor and Pinkas [19,21], who studied the problem under the different name of 1-out-of-N oblivious transfer (OT_1^N).

The situation, however, is not the same for most of the generalized variants of SPIR. A number of generalizations and extensions to SPIR have been studied in the literature. Examples include private keyword search [6,9], private element rank [8], and even more generally, the problem of oblivious decision tree, and branching program evaluation [13,4]. The existing solutions for these problems often require a number of public-key operations that is proportional to the server's input size and hence are not computationally efficient for use in practice. The only exception (with a small number of asymmetric operations) is Yao's garbled circuit protocol which is unsuitable for our applications due to its high computational cost for the client (see the related work section for a more detailed discussion).

Why OT extension does not solve the problem. OT extension techniques (e.g. [12]) are often used to reduce the number of asymmetric operations in crypto-

[1] The experimental studies we cite here, focus on PIR but the implications are even more valid for SPIR schemes.

graphic constructions. They allow one to reduce the computation needed for a large number (n) of 1-out-of-2 OTs, to k such OTs, and $O(n)$ symmetric-key operations, where k is the security parameter. This yields significant savings when n is large. One may wonder whether similar techniques can be applied to the existing solutions for the problems we are studying in order to reduce their computation. Specifically, the constructions of [15,13] can be seen as evaluation of many OTs which makes them suitable candidates for OT extension. These constructions, however, require additional properties from the underlying OT such as (i) *short OT answers* since the OT protocol is applied in multiple layers and (ii) a *strongness property* which requires the OT answer not to reveal the corresponding OT query (see future sections for more detail). Unfortunately, the existing OT extension techniques do not preserve either one of these properties and hence cannot be used to improve the computational efficiency of these solutions. Designing new extension techniques that preserves the above two properties is, however, an interesting research question.

In this work, we propose new and efficient protocols for oblivious tree and branching program evaluation which possess the following four efficiency properties:

- The number of exponentiations by both the client and the server is independent of the size of the server's input.
- The client's total computation is independent of the size of the server's input.
- Our protocols are black-box constructions based on OT_1^2 and a PRG, and hence can be instantiated using a number of assumptions.
- The protocols are non-interactive (one round) if the underlying OT_1^2 is.

RAM model of computation. When measuring client's computation in our protocols, we work in the RAM model of computation where lookups can be performed in constant time. In particular, even though the server communicates a somewhat large (proportional to the size of the program) encrypted decision program to the client, client only needs to lookup a small number of values and perform computation only on those values.

Next, we review our protocols in more detail.

1.1 Overview of Protocols

Oblivious evaluation of trees. Our first protocol deals with secure evaluation of arbitrary decision trees that are publicly known by both the client and the server, but where the input to the decision tree is only known to the client and the labels of the terminal nodes are only known to the server. This problem has a number of interesting applications. For example, 1-out-of-N oblivious transfer can be seen as an instance of this more general problem. In fact, our protocol yields a new and more efficient 1-out-of-N OT, that reduces the number of symmetric-key operations needed by the scheme of [19] from $O(N \log N)$ to $O(N)$, while maintaining the same asymmetric ($O(\log N)$) and communication ($O(kN)$) complexities.

Hiding the tree structure. Our first protocol mentioned above hides the leaf labels but assumes that the decision tree itself is public. We apply a number of additional tricks to hide all the structural information about the decision tree (except for its size), without increasing the computational cost for the client or the server. Once again, the resulting protocol preserves the above-mentioned efficiency properties. Unlike our first protocol, for this construction we need a OT_1^2 protocol with the slightly *stronger security* property that, the OT answers are not correlated with their corresponding queries. This notion of security for OT and its instantiation based on standard assumptions has already been studied by Ishai and Paskin [13] (see section 2.3).

Extension to branching programs. Finally, at the cost of a slightly higher number of OTs (though still independent of the size of the program), we extend the protocol from decision trees to decision programs (branching programs). The difficulty is to make sure the number of occurrences of a single variable during the evaluation of an input by the program is not revealed to the client. For decision trees, this number is always one, but for decision programs, it can be an arbitrary value.

Our protocols all follow the common paradigm of having the server encrypt his database/decision tree/branching program using a set of random strings; sending it to the client; and then engaging in a small number of OT_1^2 protocols such that the client learns enough keys to evaluate his input on the encrypted program and learn the output but nothing else. The main challenge is to devise an encryption strategy that is secure and at the same time allows our protocols to have the efficiency properties we are after.

We prove our protocols secure in the ideal/real world simulation paradigm. We also discuss how our new protocols yield computationally efficient constructions for a number of well-studied problems in the literature such as 1-out-of-N OT and private keyword search.

1.2 Related Work

There are a number of works that study the problem of oblivious decision program evaluation. In [13], a one-round protocol for oblivious branching program (BP) evaluation is proposed. This protocol hides the size of the BP as well as its structure. Although the number of client exponentiations are $O(n)$ and hence proportional to the size of its input, the number of exponentiations by the server is linear in the size of the BP which makes the protocol computationally quite expensive. This protocol was slightly improved in [17], where a more communication efficient protocol but with the same computational complexity is designed.

In [4] Yao's garbled circuit protocol is used in conjunction with homomorphic encryption and oblivious transfer to solve the problem of oblivious BP evaluation (with the application of remote diagnostic programs). This protocol has $O(|V|)$ rounds and requires $O(|V| + n)$ exponentiations on the client side and $O(|V|)$ exponentiations at server side where $|V|$ is the size of the program and n is the size of client's input. This protocol was later generalized and improved in

[2] but the number of server's exponentiation is still dependent on $|V|$. These constructions, however, consider a more general form of BP where the decision nodes contain an attribute index as well as a threshold value which is used to decide whether to go left or right n next.

Using fully homomorphic encryption schemes. The problem of oblivious tree/branching program evaluation can also be solved using the recent fully homomorphic encryption schemes [11]. The problem with such a solution is its high computation cost as the number of times the corresponding public-key encryption scheme is invoked is at least linear in the tree/branching program size and its input.

Using yao's garbled circuit protocol. It is also possible to use Yao's garbled circuit protocol to implement oblivious tree and branching program evaluation. One party's input to the circuit is his input string while the other party's input is the tree/branching program itself. However, Yao's protocol is not well-suited for the client/server model of computation since both parties have to perform work that is proportional to the size of the circuit, and the circuit in this case is at least linear in size of the program, and its input. In particular, in Yao-based solutions, the client ends up doing work that is proportional to the size of the server's input which does not meet our efficiency criteria.

We give a more detailed comparison of efficiency between our protocol and the existing solutions including the one based on Yao's garbled circuit protocol in Table 1.

2 Preliminaries

In this section, we introduce the notations, decision program definitions and the primitives we use throughout the paper. Readers can refer to the full version [18] for the security definitions we work with.

2.1 Notations

We denote by $[n]$ the set of positive integers $\{1, \ldots, n\}$. We use $\overset{\$}{\leftarrow}$ to denote generation of uniformly random strings.

Throughout the paper, we use k to denote the security parameter. We denote an element at row i and column j of a matrix by $M[i, j]$. Vectors are denoted by over-arrowed lower-case letters such as $\boldsymbol{v}$. We use $a||b$ to denote the concatenation of the strings a and b.

We denote a random permutation function by $Perm$. $\boldsymbol{v} \leftarrow Perm(V)$ takes as input a set of integers $V = \{1, \ldots, |V|\}$, permutes the set uniformly at random and returns the permuted elements in a row vector $\boldsymbol{v}$ of dimension $|V|$.

2.2 Decision Trees and Branching Programs

Below we define decision trees and branching programs, two common models for computation which are also the main focus of this paper. Note that we only give one definition below for both models under the name of *decision programs*. If the directed acyclic graph we mention below is a tree, then we have a decision tree and otherwise we have a branching program. The description and the notations are mostly borrowed from [13].

Definition 1 (Decision Program (DP)). *A (deterministic) decision program over the variables $x = (x_1, \ldots, x_n)$ with input domain I and output domain O is defined by a tuple $(G = (V, E), v_1, T, \psi_V, \psi_T, \psi_E)$ where:*

- *G is a directed acyclic graph (e.g. a binary tree as a special case). Denote by $\Gamma(v)$ the children set of a node v.*
- *v_1 is an initial node of indegree 0 (the root in case of a tree). We assume without loss of generality that every $u \in V - \{v_1\}$ is reachable from v_1.*
- *$T \subseteq V$ is a set of terminal nodes of outdegree 0 (the leaves in case of a tree).*
- *$\psi_V : V - T \to [n]$ is a node labeling function assigning a variable index from $[n]$ to each nonterminal node in $V - T$.*
- *$\psi_T : T \to O$ is a node labeling function asigning an output value to each terminal node in T.*
- *$\psi_E : E \to 2^I$ is an edge labeling function, such that every edge is mapped to a non-empty set, and for every node v the sets labeling the edges to nodes in $\Gamma(v)$ form a partition of I.*

In this paper, for simplicity, we describe our protocols for *binary decision programs*. But, it is easy to generalize our constructions to t-ary decision protocols for arbitrary positive integers t.

Definition 2 (Binary DP). *A binary decision program is simply formed by considering $I = \{0, 1\}$. Also for simplicity instead of the children set function Γ, we define $\Gamma_L(v)$ and $\Gamma_R(v)$ which output the variable indices of the left and right children of v. Since edge labeling are fairly obvious for binary decision programs, we often drop ψ_E when discussing such programs.*

Definition 3 (Layered DP). *We say that $P = (G = (V, E), v_1, T, \psi_V, \psi_T, \psi_E, \psi_\ell)$ is a* layered *decision program of length ℓ if the node set V can be partitioned into $\ell + 1$ disjoint levels $V = \cup_{i=0}^{\ell} V_i$, such that $V_1 = \{v_1\}, V_{\ell+1} = T$, and for every $e = (u, v)$ we have $u \in V_i, v \in V_{i+1}$ for some i. We refer to V_i as the i-th level of P. Note that we also introduced the function $\psi_\ell : V \to [\ell]$ which takes a vertex as input and returns its level as output.*

How to evaluate a DP. The output $P(x)$ of a decision program P on an input assignment $x \in I^n$ is naturally defined by following the path induced by x from v_1 to a terminal node v_ℓ, where the successor of node v is the unique node v' such that $x_{\psi_V(v)} \in \psi_E(v, v')$. The output is the value $\psi_T(v_\ell)$ labeling the terminal node (leaf node) reached by the path.

Parameters of a DP. Let $P = (G = (V, E), v_1, T, \psi_V, \psi_T, \psi_E)$ be a decision program. The size of P is $|V|$. The *height* of a node $v \in V$, denoted *height*(v), is the length (in edges) of the longest path from v to a node in T. The *length* of P is the height of v_1.

2.3 Oblivious Transfer

Our protocols use Oblivious Transfer (OT) as a building block. Since we focus on protocols that run in a single round, we describe an abstraction for one-round OT protocols here [13]. A one-round OT involves a server holding a list of t secrets $(s_1, s_2, \ldots, s_t)$, and a client holding a selection index i. The client sends a query q to the server who responds with an answer a. Using a and its local secret, the client is able to recover s_i.

More formally, a one-round 1-out-of-t oblivious transfer (OT_1^t) protocol is defined by a tuple of PPT algorithms $OT_1^t = (\mathrm{G_{OT}}, \mathrm{Q_{OT}}, \mathrm{A_{OT}}, \mathrm{D_{OT}})$. The protocol involves two parties, a client and a server where the server's input is a t-tuple of strings $(s_1, \ldots, s_t)$ of length τ each, and the client's input is an index $i \in [t]$. The parameters t and τ are given as inputs to both parties. The protocol proceeds as follows:

1. The client generates $(pk, sk) \leftarrow \mathrm{G_{OT}}(1^k)$, computes a query $q \leftarrow \mathrm{Q_{OT}}(\mathrm{pk}, 1^t, 1^\tau, \mathrm{i})$, and sends (pk, q) to the server.
2. The server computes $a \leftarrow \mathrm{A_{OT}}(\mathrm{pk}, \mathrm{q}, \mathrm{s_1}, \ldots, \mathrm{s_t})$ and sends a to the client.
3. The client computes and outputs $\mathrm{D_{OT}}(\mathrm{sk}, \mathrm{a})$.

In the case of semi-honest adversaries many of OT protocols in the literature are one-round protocols [1,20,14]. In case of malicious adversaries (CRS model), one can use the one-round OT protocols of [23].

Strong Oblivious Transfer. When OT is invoked multiple times as a sub-protocol, sometimes it is crucial for the security of the protocol that the receiver (i.e. client) be unable to correlate OT answers with their corresponding queries. In particular, when the client receives an OT answer, he should not determine which OT query the answer belongs to.

This property can be formalized by requiring the distribution of the answer a conditioned on the output s_i to be independent of the query q. More formally,

Definition 4 (Strong OT Property [13]). *An OT protocol is said to have the strong OT property if there exists an expected polynomial time simulator* $\mathrm{Sim_{OT}}$ *such that the following holds. For every $k, t, \tau, i \in [t]$, pair (pk, q) that can be generated by* $\mathrm{G_{OT}}$,$\mathrm{Q_{OT}}$ *on inputs k, t, τ, i, and strings $s_0, ..., s_{t-1} \in \{0,1\}^\tau$, the distributions* $\mathrm{A_{OT}}(\mathrm{pk}, \mathrm{q}, \mathrm{s_1}, ..., \mathrm{s_t})$ *and* $\mathrm{Sim_{OT}}(\mathrm{pk}, 1^t, \mathrm{s_i})$ *are identical.*

Some implementations of one-round OT based on homomorphic encryption schemes [15,13] satisfy this strongness property.

3 Secure Evaluation of Binary Decision Trees

In this section we propose a new protocol for secure evaluation of any publicly known decision tree with privately held terminal nodes on private inputs. This problem has a number of interesting applications such as an improved OT_1^N protocol which is described in Section 6.

Protocol Overview. Our first protocol deals with secure evaluation of arbitrary decision trees ($P = ((V,E), v_1, T, \psi_V)$) that are publicly known by both the client and the server, but where the input to the decision tree ($X = x_1x_2 \ldots x_n \in \{0,1\}^n$) is only known to the client and the labels of the terminal nodes (ψ_T) are only known to the server.

The Protocol 1

Shared Inputs: The security parameter k, and a binary decision tree $P = ((V,E), v_1, T, \psi_V)$ with $O = \{0,1\}^k$ (note the missing ψ_T). Parties also agree on a 1-out-of-2 OT protocol $(\mathrm{G_{OT}}, \mathrm{Q_{OT}}, \mathrm{A_{OT}}, \mathrm{D_{OT}})$ and a PRG $G : \{0,1\}^k \rightarrow \{0,1\}^{2k}$.

Server's Input: The terminal node labeling function ψ_T.

Client's Input: A bitstring $X = x_1x_2 \ldots x_n \in \{0,1\}^n$.

1. **The client encrypts his inputs using OT queries, and sends them to the server.**

```
Client computes $(pk, sk) \leftarrow \mathrm{G_{OT}}(1^k)$
for $1 \leq i \leq n$ do
    Client computes $q_i \leftarrow \mathrm{Q_{OT}}(pk, 1^2, 1^k, x_i)$
end for
Client sends $pk$ and $\boldsymbol{q} = (q_1, q_2, \ldots, q_n)$ to Server.
```

2. **Server computes the OT answer vector $\boldsymbol{a}$.**

```
for $1 \leq i \leq n$ do
    $(K_i^0, K_i^1) \xleftarrow{\$} \{0,1\}^k$
    $a_i \leftarrow \mathrm{A_{OT}}(pk, q_i, K_i^0, K_i^1)$
end for
$\boldsymbol{a} \leftarrow (a_1, a_2, \ldots, a_n)$
```

3. **Server prepares the Encrypted Vertex Vector $\overrightarrow{EVV}$.**

```
- Server generates a random pad vector $PAD$ of length $|V|$:
    for $i = 1$ to $|V|$ do
        $PAD[i] \xleftarrow{\$} \{0,1\}^k$
    end for
- Server encrypts the non-terminal nodes:
    for $i \in V - T$ do
        $Enc_L = K^0_{\psi_V(i)} \oplus PAD[\Gamma_L(i)]$
        $Enc_R = K^1_{\psi_V(i)} \oplus PAD[\Gamma_R(i)]$
        $EVV[i] \leftarrow G(PAD[i]) \oplus (Enc_L || Enc_R)$
    end for
- Server encrypts the labels of the terminal nodes:
    for $i \in T$ do
        $EVV[i] \leftarrow PAD[i] \oplus \psi_T(i)$
    end for
```

4. **Server sends ($\boldsymbol{a}$, $PAD[1]$, $\overrightarrow{EVV}$) to the client.**
5. **Client retrieves the keys and computes the final output.**

```
node ← 1
pad ← $PAD[1]$
while $node \notin T$ do
    $Enc_L || Enc_R \overset{parse}{\longleftarrow} EVV[node] \oplus G(pad)$
    $i \leftarrow \psi_V(node)$
    $K_i^{x_i} \leftarrow \mathrm{D_{OT}}(sk, a_i)$
    if $(x_i = 0)$ then
        $newpad \leftarrow K_i^0 \oplus Enc_L$
        $newnode \leftarrow \Gamma_L(node)$
    else
        $newpad \leftarrow K_i^1 \oplus Enc_R$
        $newnode \leftarrow \Gamma_R(node)$
    end if
    $pad \leftarrow newpad$
    $node \leftarrow newnode$
end while
Client outputs $(pad \oplus EVV[node])$ as his final output.
```

In our protocol, a pair of random keys $(K^0_{x_i}, K^1_{x_i})$ is generated for each x_i, and is used by the server as his input in the n 1-out-of-2 OTs. The idea is then to generate a set of random pads, one for each node in the decision tree. Each node stores a pair of values, i.e. the two random pads corresponding to its left and right children. However, this pair of values is not stored in plaintext. Instead, the left (right) component of the pair is encrypted using a combined key formed by XORing the left-half (right-half) of the expanded pad (expanded using a PRG) for the current node with $K^0_{x_i}$ ($K^1_{x_i}$) where i is the label of the current node. The encryption scheme is a simple one-time pad encryption. The encrypted values are stored in a vector we call the Encrypted Vertex Vector ($\overrightarrow{EVV}$).

The client who receives one of each pair of random keys, can then use them to decrypt a single path on the tree corresponding to the evaluation path of his input X, and recover his output, i.e. the output label associated with the reached terminal node. As we show in the proof, the rest of the terminal node labels remain computationally hidden from the client. A detailed description of the protocol is depicted in the box for the protocol 1.

Security. In the full version of the paper [18] we show that as long as the oblivious transfer protocol used is secure even when executed in parallel, so is our construction given above. Particularly, if the OT is secure against malicious (semi-honest) adversaries (when run in parallel), protocol 1 described above is also secure against malicious (semi-honest) adversaries. The following Theorem formalizes this statement.

Theorem 1. *In the OT-hybrid model, the above protocol is* fully-secure *(i.e. simulation-based security: see the definition in Appendix B of the full version [18]) against malicious adversaries.*

Complexity. The proposed protocol runs in one round which consists of a message from the client to the server and vice versa.

The only asymmetric computation required in this protocol is for the OT invocations and since there are n OT invocations and each OT requires a constant number of exponentiations, the number of exponentiations is $O(n)$ for both parties. Using the OT extension of [12] we can reduce the number of exponentiations to $O(k)$ for both parties.

The number of other (symmetric-key) operations such as PRG invocations, and XORing is $O(|V|)$ on the server side and $O(l)$ on the client side where l refers to the depth of the tree ($l \leq n$).

The communication complexity of the protocol is dominated by the total size of the elements in $\overrightarrow{EVV}$ which is bounded by $O(|V|k)$ where k is the security parameter. This is due to the fact that each element of $\overrightarrow{EVV}$ is of size $2k$ and there are $|V|$ such elements.

4 Hiding the Tree Structure

We will show how to securely formulate via decision programs, other protocols such as the *private keyword search* problem [6,9] and the *private element rank* problem [8] (see the full version for discussion on the latter). For some of these problems, privacy of the server's database critically relies on keeping the structure of the corresponding decision program private. The program structure includes all the information available about it except for its size (number of its nodes) and the number of variables (both of which are publicly known).

For simplicity, in this section we assume that the decision tree we work with is layered. Alternatively, we could allow for arbitrary tree structures[2] and then consider the length of the evaluation path as public information available to our

[2] In a non-layered decision tree, the length of the evaluation path for different inputs need not be the same.

simulators (our protocol reveals the length). However, since in most applications one wants to keep this information private (see Section 6), we chose to work with this assumption instead. Note that there are generic and efficient ways of transforming any decision program (tree) into a layered one. In section 6, we give a customized and more efficient transformation for the private keyword search application.

Next, we show how to enhance the protocol of previous section in order to hide the decision tree's structure without increasing the computational cost of the client or the server (in the next section we extend this to decision programs). Once again, our protocol reduces the problem to n OT_1^2 protocols.

Here we require the OT_1^2 protocol to have a slightly stronger security property compared to the standard one. We refer to such OTs as *strong* OTs. At a high level, we require that the OT answers do not reveal anything about the corresponding query. This property helps us hide from the client, the order in which the input variables are evaluated which would in part reveal some information about the structure of the tree. A formal definition of security as well as some existing constructions for strong OT are discussed in section 2.3.

An Overview. The high level structure of the protocol of this section is similar to the previous one. In particular, we still perform n OTs and use a set of key pairs and random pads in order to garble the tree. But since this time we are also interested in hiding the structure of the tree, a more involved encryption process is necessary. The main changes to the previous construction are as follows: first, instead of revealing the labels of the non-terminal nodes to the client, we use a pointer (index) to the corresponding item in the *randomly permuted list of OT answers* ($\boldsymbol{a'}$). In order for the permuted list of answers not to reveal the permutation, we need to use a *strong* OT protocol. Second, in order to hide the arrangement of the nodes in the tree, instead of revealing the outgoing edges of each non-terminal node, we use two pointers to the corresponding nodes in a *randomly permuted list of the nodes* in the tree ($\overrightarrow{EVV}$).

The three pointers mentioned above (one pointing to $\boldsymbol{a'}$ and two pointing to $\overrightarrow{EVV}$) stored at each node, is all that the client needs in order to evaluate the decision tree on his input. All of this information will be encrypted using a combination of the random pads and the key pairs similar to the construction of previous section and is stored in the $\overrightarrow{EVV}$ vector. However, several subtleties exist in order to make sure the construction works. First, only the random pads (not the random keys) are to be used in encrypting the pointers to the OT answers since the random keys themselves are retrieved from the OT answers. Also, in order to hide from the client which bit value the retrieved key corresponds to (note that this can reveal extra information about the node labels which are to be kept private), the two values encrypted using the keys (Enc_L and Enc_R) are randomly permuted and a redundant padding of 0^k is appended to the values before encryption to help the client recognize when the correct value is decrypted. A detailed description of the protocol is depicted in the box for protocol 2.

In the description above, the size of EVV for terminal vs. non-terminal nodes is different which leaks the total number of terminal nodes. However, we only did so to make the description of the protocol simpler. In particular, it is easy to pad the size of terminal nodes to the appropriate size, and then embed an indicator bit in each EVV cell (before encryption) that helps the client determine if he has reached a terminal node.

The Protocol 2

Shared Inputs: The security parameter k, size of the set V, i.e. $|V|$. We also let $k' = 2k + \log(|V|)$. Parties also agree on a *strong* OT_1^2 protocol $OT = (\mathrm{G_{OT}}, \mathrm{Q_{OT}}, \mathrm{A_{OT}}, \mathrm{D_{OT}})$ and a PRG $G : \{0,1\}^k \rightarrow \{0,1\}^{2k'+\log n}$.

Server's Input: A layered binary decision tree $P = ((V, E), v_1, T, \psi_V, \psi_T)$ with $O = \{0,1\}^k$.

Client's Input: A bitstring $X = x_1 x_2 \ldots x_n \in \{0,1\}^n$.

1. **Client encrypts his inputs using OT queries, and sends the vector $\boldsymbol{q}$ to Server.** The first step of computing the OT queries for the client is identical to protocol of Section 3 and hence is omitted here.
2. **Server computes a permuted OT answer vector $\boldsymbol{a'}$.**
   - Server computes the OT answer vector $\boldsymbol{a}$:

```
for $1 \le i \le n$ do
    $(K_i^0, K_i^1) \xleftarrow{\$} \{0,1\}^{k'}$
    $a_i \leftarrow \mathrm{A_{OT}}(pk, q_i, K_i^0, K_i^1)$
end for
$\boldsymbol{a} \leftarrow (a_1, a_2, \ldots, a_n)$
```

   - Server Generates a random permutation vector PER_n:

```
$PER_n \leftarrow Perm(\{1, ..., n\})$
```

   - Server Permutes $\boldsymbol{a}$ using PER_n:

```
for $1 \le i \le n$ do
    $a'[PER_n[i]] \leftarrow a[i]$
end for
```

3. **Server computes an encrypted vertex vector $\overrightarrow{EVV}$.**
   - Server generates a random pad vector PAD of length $|V|$:

```
for $i = 1$ to $|V|$ do
    $PAD[i] \xleftarrow{\$} \{0,1\}^k$
end for
```

   - Server Generates a random permutation vector PER_V:

```
$PER_V \leftarrow Perm(\{1, ..., |V|\})$
```

   - Server encrypts non-terminal nodes and their outgoing edges:

```
for $i \in V - T$ do
    $Enc_L \leftarrow K^0_{\psi_V(i)} \oplus$
            $(PAD[\Gamma_L(i)] || PER_V[\Gamma_L(i)] || 0^k)$
    $Enc_R \leftarrow K^1_{\psi_V(i)} \oplus$
            $(PAD[\Gamma_R(i)] || PER_V[\Gamma_R(i)] || 0^k)$
    $b \xleftarrow{\$} \{0,1\}$
    if $b = 0$ then
        $EVV[PER_V[i]] \leftarrow G(PAD[i]) \oplus$
            $(PER_n[\psi_V(i)] || Enc_L || Enc_R)$
    else
        $EVV[PER_V[i]] \leftarrow G(PAD[i]) \oplus$
            $(PER_n[\psi_V(i)] || Enc_R || Enc_L)$
    end if
end for
```

   - Server encrypts the labels of the terminal nodes:

```
for $i \in T$ do
    $EVV[PER_V[i]] \leftarrow PAD[i] \oplus \psi_T(i)$
end for
```

4. **Server sends $(PER_V[1]$, $PAD[1]$, $\boldsymbol{a'}$, $\overrightarrow{EVV})$ to Client.**
5. **Client retrieves the keys and computes the final result.**

```
$node \leftarrow PER_V[1]$
$pad \leftarrow PAD[1]$
while $node \notin T$ do
    $(j || Enc_0 || Enc_1) \overset{parse}{\longleftarrow} EVV[node] \oplus pad$
    $K \leftarrow D_{OT}(sk, a'[j])$
    $Dec_0 \leftarrow K \oplus Enc_0$
    $Dec_1 \leftarrow K \oplus Enc_1$
    if $k$ least significant bits of $Dec_0$ are 0 then
        $pad || node || 0^k \overset{parse}{\longleftarrow} Dec_0$
    else
        $pad || node || 0^k \overset{parse}{\longleftarrow} Dec_1$
    end if
    $pad \leftarrow G(pad)$
end while
Client outputs $(pad \oplus EVV[node])$ as his final output.
```

Security. The simulation proof for protocol 2 follows the same line of argument as that of protocol 1. The main difference in the security claim for protocol 2 is that it is *private* against a malicious server (as opposed to being fully-secure). The intuition behind this weakening in the security guarantee is that the server can construct an $\overrightarrow{EVV}$ that does no correspond to a valid decision tree, and our protocol does not provide any mechanisms for detecting this type of behavior. However, the protocol is still private, since all that the server sees in the protocol are the OT queries. Also note that the client is always able to compute an output even if the $\overrightarrow{EVV}$ is not a valid tree (there is no possibility of failure conditioned on specific input values), and hence the server cannot take advantage of the pattern of aborts by the client in order to learn additional information. It is possible to augment the protocol with zero-knowledge proofs that yield full security against a malicious server, but all the obvious ways of doing so would diminish the efficiency properties we are after. In particular both the server and the client would have to do a number of exponentiations that is proportional to the size of the tree.

Next, we state our security theorem. Readers are referred to the full version of the paper [18] for the proof of Theorem 2.

Theorem 2. *In the strong-OT-hybrid model, and given a cryptographically secure PRG G, the above protocol is* fully-secure *against a malicious client and is private against a malicious server.*

Complexity. Similar to protocol 1, protocol 2 runs in one round. The asymptotic computational complexity for the client and the server remains the same too. In other words, the client and the server perform $O(n)$ exponentiations for the OTs. Server performs $O(|V|)$ PRG invocations and XOR operations while the client performs $O(l)$ PRG and XOR operations where l is the length of the decision tree.

The communication cost of the protocol is dominated by size of $\overrightarrow{EVV}$ which consists of $|V|$ elements of size $4k + \log |V|$. This leads to a total communication of $O(|V|(\log |V| + k)$ bits.

5 Extension to Branching Programs (BP)

In this section we extend our proposed protocol of previous section to branching programs (BP). BPs are decision programs that are represented as directed acyclic graphs [25], and hence may contain various paths from the root to some nodes. Because of the structure of the BPs, a variable may be evaluated more than once in a single evaluation. In order to hide the number of times a variable is visited from a curious party, and to obliviously evaluate a BP, we generate a separate OT answer vector for each level of the BP. This can be done by computing a permuted OT answer matrix (A') instead of an OT answer vector described in protocol 2, and using the indices to this matrix when $\overrightarrow{EVV}$ is computed. Similar to the previous protocol, we need a strong OT as a sub-protocol to prevent correlation between OT queries, and answers.

More formally, assume that PER_{ln} is a permutation matrix of dimension $l \times n$ where l is the length of the program. A' is computed as follows:

```
for $i = 1$ to $l$ do
  for $j = 1$ to $n$ do
    $(K^0_{i,j}, K^1_{i,j}) \xleftarrow{\$} \{0,1\}^{k'}$
    $A'[i, PER_{ln}[i,j]] \leftarrow A_{OT}(pk, q_j, K^0_{i,j}, K^1_{i,j})$
  end for
end for
```

Moreover, in the $\overrightarrow{EVV}$ computation step, $PER_{ln}[height(i), \psi_V(i)]$ is used to point to the elements in A'. To compute the final result, the client will also use the elements in the A' matrix to retrieve the keys and evaluate the BP.

The argument for the security of this scheme is almost the same as protocol 2.

Theorem 3. *In the strong-OT-hybrid model, and given a cryptographically secure PRG G, the above-mentioned protocol is* fully-secure *against a malicious client and is* private *against a malicious server.*

Complexity. As before, the protocol runs in one round. The number of exponentiations performed by the client remains the same but the server has to perform slightly more exponentiations. In other words, the client performs $O(n)$ exponentiations and the server performs $O(ln)$ exponentiations for the OTs where l is the length of the branching program. The number of PRG invocations and XOR operations remains the same as protocol 2 which is $O(|V|)$ for the server and $O(l)$ for the client. The asymptotic communication cost of the protocol remains the same as protocol 2 which is $O(|V|(\log|V| + k))$ bits.

Table 1 compares the complexities of the related work with our proposed protocol for oblivious evaluation of BPs. The main advantage of our proposed protocol over the previous schemes is that the server's asymmetric computation is independent of the size of the branching program. This feature makes our protocol truly efficient when $|V|$ is large. In case of Yao-based constructions, the size of the circuit required for evaluating a branching program of size $|V|$ and length l on an input of size n is $O(|V|l(\log|V| + \log n))$. Therefore, using Yao's protocol for oblivious branching program evaluation yields a protocol which needs $O(|V|l(\log|V| + \log n))$ symmetric-key operations for both the client and the server, and a communication complexity of $O(|V|lk(\log|V| + \log n))$.

Table 1. Comparison of protocols for oblivious branching program evaluation

	Rounds	Client Computations		Server Computations		Communication Complexity
		Asymmetric	Symmetric	Asymmetric	Symmetric	
Yao [26]	1	$O(n)$	$O(\|V\|l(\log\|V\| + \log n))$	$O(n)$	$O(\|V\|l(\log\|V\| + \log n))$	$O(\|V\|lk(\log\|V\| + \log n))$
[13]	1	$O(n+l)$	none	$O(\|V\|)$	none	$O(knl)$
[4,2]	$O(\|V\|)$	$O(\|V\| + n)$	$O(\|V\|)$	$O(\|V\|)$	$O(\|V\|)$	$O(k(n + \|V\|))$
Ours	1	$O(n)$	O(l)	$O(ln)$	$O(\|V\|)$	$O(\|V\|(\log\|V\| + k))$

6 Applications

An Improved OT^N_1 Protocol. We review the Naor-Pinkas OT^N_1 and its efficiency in Appendix C of [18]. It is easy to observe that looking up an index

$X = x_1 \cdots x_{\log N}$ in a database of size N can be efficiently described as evaluation of a decision tree on X, where the node variables are x_i's and the terminal (leaf) node values are the elements of the database. This way, an OT_1^N can be represented as a special case of our proposed protocol 1. This yields a more efficient OT_1^N protocol with only $O(N)$ instead of $O(N \log N)$ PRG invocations which is the case in the Naor-Pinkas protocol [19].

Claim 1. *Let N be the size of the database. Given a one-round 1-out-of-2 OT protocol with security against malicious adversaries, there exists a one-round two-party protocol for 1-out-of-N OT, with full-security against malicious adversaries. The protocol only requires $O(\log N)$ exponentiations by both parties. The total work of the client is $O(\log N)$, while the server performs $O(N)$ PRG invocations.*

The security of the construction follows from our more general construction in Section 3. It is also easy to verify the claimed computational complexities.

A Private Keyword Search Protocol. We first recall the setup for the private keyword search (PKS) problem. A server and a client are involved in this problem. The server's input is a database D of N pairs (k_i, p_i), where $k_i \in \{0,1\}^\ell$ is a keyword, and $p_i \in \{0,1\}^m$ is the corresponding payload. The client's input is an ℓ bit searchword $w = w_1 w_2 \cdots w_\ell$. If there is a pair (k_i, p_i) in the database such that $k_i = w$, then the output is the corresponding payload p_i. Otherwise the output is a special symbol $\perp$.

Designing efficient PKS protocols has been the focus of several works in the literature [6,9]. However, these works have mostly focused on optimizing the communication complexity of the protocols. In particular, they require $O(N)$ exponentiations by the server, which is a significant computational burden for large N.

Using the techniques we developed in previous section, we can design an efficient PKS with properties mentioned in the claim below. The details of the construction are available in the full version of the paper [18].

Claim 2. *Let ℓ be the length of the keywords and N be the size of the database. Given a one-round OT protocol with security against malicious adversaries, there exists a one-round two-party protocol for private keyword search, with full-security against a malicious client and privacy against a malicious server. The protocol only requires $O(\ell)$ exponentiations by both parties. The total work of the client is $O(\ell)$, while the server performs $O(N\ell)$ symmetric operations.*

References

1. Aiello, B., Ishai, Y., Reingold, O.: Priced Oblivious Transfer: How to Sell Digital Goods. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 119–135. Springer, Heidelberg (2001)
2. Barni, M., Failla, P., Kolesnikov, V., Lazzeretti, R., Sadeghi, A.-R., Schneider, T.: Secure Evaluation of Private Linear Branching Programs with Medical Applications. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 424–439. Springer, Heidelberg (2009)

3. Beimel, A., Ishai, Y., Malkin, T.: Reducing the Servers Computation in Private Information Retrieval: PIR with Preprocessing. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 55–73. Springer, Heidelberg (2000)
4. Brickell, J., Porter, D., Shmatikov, V., Witchel, E.: Privacy-preserving remote diagnostics. In: ACM CCS 2007, pp. 498–507 (2007)
5. Cachin, C., Micali, S., Stadler, M.: Computationally Private Information Retrieval with Polylogarithmic Communication. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 402–414. Springer, Heidelberg (1999)
6. Chor, B., Gilboa, N., Naor, M.: Private information retrieval by keywords (1997) (manuscript)
7. Chung, K.-M., Kalai, Y., Vadhan, S.: Improved Delegation of Computation Using Fully Homomorphic Encryption. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 483–501. Springer, Heidelberg (2010)
8. Dedic, N., Mohassel, P.: Constant-Round Private Database Queries. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 255–266. Springer, Heidelberg (2007)
9. Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword Search and Oblivious Pseudorandom Functions. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 303–324. Springer, Heidelberg (2005)
10. Gennaro, R., Gentry, C., Parno, B.: Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 465–482. Springer, Heidelberg (2010)
11. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: ACM STOC 2009, pp. 169–178 (2009)
12. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending Oblivious Transfers Efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003)
13. Ishai, Y., Paskin, A.: Evaluating Branching Programs on Encrypted Data. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 575–594. Springer, Heidelberg (2007)
14. Kalai, Y.T.: Smooth Projective Hashing and Two-Message Oblivious Transfer. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 78–95. Springer, Heidelberg (2005)
15. Kushilevitz, E., Ostrovsky, R.: Replication is not needed: Single database, computationally-private information retrieval. In: FOCS 1997, pp. 364–373 (1997)
16. Lipmaa, H.: An Oblivious Transfer Protocol with Log-Squared Communication. In: Zhou, J., López, J., Deng, R.H., Bao, F. (eds.) ISC 2005. LNCS, vol. 3650, pp. 314–328. Springer, Heidelberg (2005)
17. Lipmaa, H.: Private branching programs: On communication-efficient cryptocomputing. Tech. rep., Cryptology ePrint Archive, Report 2008/107 (2008)
18. Mohassel, P., Niksefat, S.: Oblivious decision programs from oblivious transfer: Efficient reductions (full version) (2011), `http://pages.cpsc.ucalgary.ca/~pmohasse/odp.pdf`
19. Naor, M., Pinkas, B.: Oblivious transfer and polynomial evaluation. In: ACM STOC 1999, pp. 245–254. ACM (1999)
20. Naor, M., Pinkas, B.: Efficient oblivious transfer protocols. In: ACM SIAM 2001, pp. 448–457 (2001)
21. Naor, M., Pinkas, B.: Computationally secure oblivious transfer. Journal of Cryptology 18(1), 1–35 (2005)

22. Olumofin, F., Goldberg, I.: Revisiting the Computational Practicality of Private Information Retrieval. In: Danezis, G. (ed.) FC 2011. LNCS, vol. 7035, pp. 158–172. Springer, Heidelberg (2012)
23. Peikert, C., Vaikuntanathan, V., Waters, B.: A Framework for Efficient and Composable Oblivious Transfer. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 554–571. Springer, Heidelberg (2008)
24. Sion, R., Carbunar, B.: On the computational practicality of private information retrieval. In: NDSS 2007, pp. 2006–06 (2007)
25. Sipser, M.: Introduction to the Theory of Computation. International Thomson Publishing (1996)
26. Yao, A.: Protocols for secure computations. In: FOCS 1982, pp. 160–164 (1982)