Evaluating Branching Programs on Encrypted Data*

Yuval Ishai and Anat Paskin

Computer Science Department, Technion
yuvali@cs.technion.ac.il, anps83@gmail.com

Abstract. We present a public-key encryption scheme with the following properties. Given a branching program P and an encryption c of an input x, it is possible to efficiently compute a *succinct* ciphertext c' from which $P(x)$ can be efficiently decoded using the secret key. The size of c' depends polynomially on the size of x and the *length* of P, but does not further depend on the size of P. As interesting special cases, one can efficiently evaluate finite automata, decision trees, and OBDDs on encrypted data, where the size of the resulting ciphertext c' does not depend on the size of the object being evaluated. These are the first general representation models for which such a feasibility result is shown. Our main construction generalizes the approach of Kushilevitz and Ostrovsky (FOCS 1997) for constructing single-server Private Information Retrieval protocols.

We also show how to strengthen the above so that c' does not contain additional information about P (other than $P(x)$ for some x) even if the public key and the ciphertext c are maliciously formed. This yields a two-message secure protocol for evaluating a length-bounded branching program P held by a server on an input x held by a client. A distinctive feature of this protocol is that it hides the size of the server's input P from the client. In particular, the client's work is independent of the size of P.

1 Introduction

Computing on encrypted data is arguably one of the most intriguing open problems in cryptography. The variant of this problem we are interested in may be illustrated by the following motivating scenario. Suppose that a client, holding a sensitive local input x, wishes to run a remote program P on this input. For instance, x can be the medical history of an individual and P a complex propriety algorithm determining whether to offer insurance coverage to this individual. To the end of evaluating $P(x)$, the client wishes to publish an *encrypted* version of x, denoted by c, while still allowing a server owning P to effectively run its program on the ciphertext c. That is, based on P and c the server should compute in polynomial time a message c', from which the client can recover $P(x)$ using its secret key.

As described so far, the problem can be solved by simply letting c' include a complete description of P. However, this trivial solution has two significant weaknesses. First, it completely reveals P to the client, whereas ideally the client should only be able to learn $P(x)$. Second, when the description size of P is larger than its input and

* Supported by grants 36/03 and 1310/06 from the Israel Science Foundation and grant 2004361 from the U.S.-Israel Binational Science Foundation.

output, this solution is wasteful in terms of communication. Ideally, the communication should be *a-priori* bounded by some polynomial in the size of the input x, the output $P(x)$ and the security parameter, independently of the description size of P. The same holds for the amount of local computation and storage used by the client. To summarize, it is desirable to obtain solutions which satisfy the following two goals:

1. Hide P from the client (to the extent possible).
2. Make the client's work independent of the size of P. In particular, c' should be *succinct* in the sense that its size depends only on the size of the input and output and not on that of P.

Jumping ahead, the main open problem in the area is that of realizing the second goal. This problem is the focus of our work.

Before addressing known methods for realizing the above two goals, it is instructive to further clarify what we mean when referring to a "program" P. A program is a string that represents a function, mapping an input x to an output y. To simplify the exposition, we restrict the attention to finite boolean functions $f : \{0,1\}^n \to \{0,1\}$. The correspondence between a program P and the function it represents is determined by an underlying *representation model*. Common representation models for finite functions include circuits, formulas, branching programs, OBDDs, finite automata, decision trees, and truth tables. Once the representation model is fixed, every string P has a unique interpretation as a program computing some specific function f. In this work we will be interested in *universal* representation models, in which every function f can be computed by some program P in the model. Note that all of the models in the above list are universal. However, the *complexity* of representing a function can greatly vary between the models. Circuits are the most powerful model in the list, in the sense that a program in any of the other models has an equivalent circuit of essentially the same size. On the other extreme, truth tables are the least powerful of these models, requiring a program of size 2^n for any function f. This makes truth tables useless for all but very small input lengths n.

We return to the question of realizing the above two goals. Goal 1 can be addressed by using techniques from the area of secure computation. Most notably, Yao's garbled circuit technique [36, 7, 25] can handle any *circuit* P, allowing to computationally hide all information about P other than $P(x)$ and the size of P. A similar result can be obtained for less powerful representation models, such as formulas or various kinds of branching programs, with the additional feature of keeping P information-theoretically private [35, 4, 18, 22]. However, all these techniques inherently fail with respect to Goal 2, as they require the size of c' to be comparable to the size of P. This gives rise to the following question:

> For which natural representation models can we realize Goal 2, namely evaluate an arbitrary program P on an encrypted input so that the client's work does not depend on the size of P?

A positive answer for the case of circuits (hence also for all other models) would easily follow from the existence of a completely malleable encryption scheme — one that allows to freely perform both additions and multiplications on ciphertexts. However, there is yet no candidate for an encryption scheme with this strong property.

The first protocols in which the client's work can go below the size of P were given in the context of Private Information Retrieval (PIR) [10, 23]. A single-server PIR protocol can be viewed as a protocol for evaluating a *truth table* P of size $N = 2^n$ on an encrypted input x of size n. There are such protocols in which the client's work is polynomial in n [6, 26], thus affirmatively answering the above question for the case of a truth table representation. Extensions to a *set* representation (where P lists the set of inputs on which f evaluates to 1) were given in the context of private keyword search [23, 9, 13, 30]. Recently, an efficient protocol for evaluating 2-DNF formulas and degree-2 polynomials on encrypted data was given by Boneh et al. [5].[1] The question of realizing Goal 2 for more powerful and useful representation models remained open.

1.1 Our Contribution

We obtain an affirmative answer to our main question for the case of *length-bounded branching programs*. To explain the meaning of this result, we give some background on branching programs and their complexity. A (deterministic) branching program P is defined by a directed acyclic graph in which the nodes are labeled by input variables and every nonterminal node has two outgoing edges, labeled by 0 and 1. An input $x \in \{0,1\}^n$ naturally induces a computation path from a distinguished initial node to a terminal node, whose label determines the output $P(x)$. The *size* of P is defined as the number of nodes in the graph and its *length* is the length of the longest path from the initial node to a terminal node. Branching programs are a relatively powerful representation model. In particular, any logarithmic space or NC^1 computation can be carried out by a family of polynomial-size branching programs.

We consider classes of branching programs whose length is bounded by some public parameter ℓ, where $\ell = \ell(n)$ is polynomial in n. Representation by $\ell(n)$-bounded branching programs is universal whenever $\ell(n) \geq n$. Indeed, any function f can be computed by a complete decision tree of length n and size $O(2^n)$. Branching programs of length $\ell(n) = n$ are of special interest, as they can simulate several representation models that are often used in practice. For instance, if f can be computed by a deterministic finite automaton with s states, then it can be computed by a branching program of length n and size $sn + 1$. Other useful models such as decision trees and OBDDs are also special cases of length-n branching programs.

Our main result is a public-key encryption scheme with the following properties. Given a branching program P and an encryption c of an input x, it is possible to efficiently compute a *succinct* randomized ciphertext c' from which $P(x)$ can be efficiently decoded using the secret key. The size of c' and the work required for decrypting it depend polynomially on the size of x and the *length* of P, but do not further depend on the size of P. Thus, whenever the length $\ell(n)$ is some fixed (polynomial) function of n, we realize Goal 2 above. As interesting special cases, one can evaluate finite automata, decision trees, and OBDDs on encrypted data, where the size of the resulting ciphertext c' does not depend on the size of the object being evaluated. These are the first general

[1] In fact, the scheme of [5] realizes a stronger form of computing on encrypted data in which the length of the ciphertext c' depends only on the security parameter and not on the length of the input.

representation models for which such a feasibility result is shown. We also strengthen the above protocol to realize Goal 1 in a very strong sense, guaranteeing that c' does not contain additional information about P (other than $P(x)$ for some x) even if the public key and the ciphertext c are maliciously formed.

Size hiding. Our protocols have the following *size hiding* feature: the ciphertext c' does not reveal any information whatsoever about the *size* of P, no matter how large P is.[2] This should be contrasted with previous methods of computing on encrypted data, in which the communication complexity and the client's work directly reflect (an upper bound on) the size of P. Thus, we achieve a stronger version of Goal 1 than in all previous solutions. A similar notion of size hiding was previously considered by Micali et al. in the context of *zero-knowledge sets* [27].

Applications to secure two-party computation. Our technique for computing on encrypted data immediately gives rise to a one-round (two-message) secure protocol for evaluating a length-bounded branching program P held by a server on an input x held by a client. (This also implies a protocol for the setting in which P is public but its inputs are partitioned between the two parties.) In the case of malicious parties, the protocol satisfies the same relaxed security definition used in previous works on one-round secure computation [29, 1, 13, 20, 24]. A distinctive feature of our protocol is that the client's work is independent of the size of P and moreover the protocol hides the size of P from the client.[3] The latter size hiding feature demonstrates that while hiding the sizes of *both* inputs is impossible for interesting functions, there are useful special cases where one can hide the size of *one* of the inputs while maintaining security.

As a concrete application, one can obtain a secure one-round protocol for *keyword search* which totally hides from the client the size of the data set held by the server. That is, a client holding a secret keyword x can query a database D held by a server without revealing x and while assuring the server that it cannot learn anything about D (including its size) other than whether $x \in D$. Previous solutions to the secure keyword search problem [9, 13, 30] fall short of achieving the size hiding goal. A size hiding protocol as above is obtained by representing D as a *trie* data structure, which can be viewed as an instance of a length-n branching program.

We finally note that the one-round protocol obtained using our technique yields a simpler alternative to similar protocols from the literature that provide *unconditional* security to the server [35, 4, 18, 22]. Its complexity improves over previous protocols even in the case of branching programs of unbounded length. For evaluating a branching program of size s over n inputs, the communication complexity of our protocol is $O(kns)$ (where k is a security parameter), improving over the $O(ks^2)$ complexity of the best previous solutions in this setting [18].

---

[2] We note that perfect size hiding cannot be achieved in the physical reality, as the *time* it takes the server to respond reveals an upper bound on the size of P. However, increasing this upper bound on the size of P does not involve additional work. This should be contrasted with the partial size hiding that can be achieved using previous protocols by simply padding the inputs.

[3] A secure two-party protocol in which the client's work is almost independent of the size of P can be obtained using the technique of Naor and Nissim [28]. However, this protocol requires multiple rounds of interaction and does not achieve size hiding.

Techniques. The basic version of our protocol uses a simple generalization of the technique of Kushilevitz and Ostrovsky [23] for constructing single-server PIR protocols. In fact, the protocol of [23] (as well as its variants from [34, 26]) can be viewed as an instance of our protocol in which the branching program is a complete (but possibly non-binary) decision tree whose i-th level depends only on the i-th input variable.

Our protocol proceeds roughly as follows. The ciphertext c is obtained by separately encrypting each bit of x using a homomorphic public-key encryption scheme. (For efficiency reasons we rely on the Damgård-Jurik scheme [11]; this scheme was previously used in the context of PIR by Lipmaa [26].) To evaluate P on x we proceed in a bottom up manner. Starting from the terminal nodes, in the i-th iteration we handle all nodes whose distance from the terminal nodes is i. For each such node, we compute a ciphertext containing an (iterated) encryption of its value. Using the homomorphic property, the encryption assigned to every node can be computed from the encryptions assigned to its children (which were computed in previous iterations) and the encryption of the input bit labeling this node. The ciphertext c' is the (iterated) encryption assigned to the initial node. The client can recover $P(x)$ by applying iterated decryptions to c'.

The stronger variant of our protocol which remains secure in the case of malicious clients is more involved, and relies on variants of previous techniques of Aiello et al. [1], Naor and Pinkas [29], Laur and Lipmaa [24], and (especially) Kalai [20].

Organization. In Section 2 we define our general notion of representation models as well as the specific branching program model for which our results apply. In Section 3 we define the problem of computing on encrypted data as well as a variant of Oblivious Transfer on which our solution relies. Our main protocol is presented in Section 4. This protocol guarantees the privacy of the client as well as the privacy of the server against a semi-honest client. The case of malicious clients is discussed in Section 5. For lack of space, some details are deferred to the full version.

2 Preliminaries

We denote by $y \leftarrow A(x)$ the process of invoking the (possibly randomized) algorithm A on input x and assigning the result to y. We say that a function $\epsilon(k)$ is negligible if for every constant $c > 1$ we have $\epsilon(k) < 1/k^c$ for all sufficiently large k. We use the following standard notion of statistical distance:

Definition 1 (Statistical distance). *Let X, Y be random variables over the finite set U. Denote the distance between X and Y by*

$$\mathsf{SD}(X,Y) = max_{U' \subseteq U} \left| \mathop{\mathbf{Pr}}_{x \leftarrow X}[x \in U'] - \mathop{\mathbf{Pr}}_{y \leftarrow Y}[y \in U'] \right|$$

2.1 Representation Models

Loosely speaking, a representation model is a way of interpreting strings as "programs" for evaluating (families of) functions over some finite domain. We are only interested in representation models which are *universal* in the sense that every function has a program evaluating it in that model. For simplicity we restrict the attention to functions defined over a binary input alphabet. An extension to the general case is straightforward.

Definition 2 (Representation model). *A* representation model *is a polynomial-time computable function $U : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$, where $U(P,x)$ is referred to as the value returned by a "program" P on the input x. When U is understood from the context, we use $P(x)$ to denote $U(P,x)$. We say that a function $f : \{0,1\}^* \to \{0,1\}^*$ can be implemented in a representation model U if there exists an infinite sequence $(P_0, P_1, \ldots)$, referred to as an implementation of f in U, such that $f(x) = U(P_{|x|}, x)$ for every $x \in \{0,1\}^*$.*

We now define the branching programs model. This is the representation model for which our main result applies.

Definition 3 (Branching program (BP)). *A (deterministic) branching program over the variables $x = (x_1, \ldots, x_n)$ with input domain I and output domain O is defined by a tuple $(G = (V,E), v_0, T, \psi_V, \psi_E)$ where:*

- *G is a directed acyclic graph. Denote by $\Gamma(v)$ the children set of a node v.*
- *v_0 is an initial node of indegree 0. We assume without loss of generality that every $u \in V - \{v_0\}$ is reachable from v_0.*
- *$T \subseteq V$ is a set of terminal nodes of outdegree 0.*
- *$\psi_V : V \to [n] \cup O$ is a node labeling function assigning an output value to each terminal node in T, and a variable index from $[n]$ to each nonterminal node in $V - T$.*
- *$\psi_E : E \to 2^I$ 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.*

BP evaluation. The output $P(x)$ of a branching program P on an input assignment $x \in I^n$ is naturally defined by following the path induced by x from v_0 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 $\psi_V(v_\ell)$ labeling the terminal node reached by the path.

BP complexity measures. Let $P = (G(V,E), v_0, T, \psi_V, \psi_E)$ be a BP. The *size* of P is $|E|$. (Note that in the case where $|I|$ is constant we have $|E| = O(|V|)$.) The *height* of a node $v \in V$, denoted $\mathsf{height}(v)$, 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_0. We say that an implementation $(P_0, P_1, \ldots)$ of a function f in the branching program model is length-bounded by $\ell(\cdot)$ if the length of each P_n is at most $\ell(n)$.

Remark 1. In the following we will sometimes assume that branching programs have binary inputs and outputs, namely that $I = O = \{0,1\}$. We stress, however, that the generalization to non-binary domains is useful for some of the applications we have in mind. For instance, non-binary input alphabets are useful for casting the PIR protocol from [23] as a special case of our main construction, and large output alphabets are useful for applications such as private retrieval by keywords [9, 13].

Our protocols take the simplest form when the branching program being evaluated is *layered* in the following sense.

Definition 4 (Layered BP). *We say that P is a* layered *branching program of length ℓ if the node set V can be partitioned into $\ell+1$ disjoint levels $V = \bigcup_{i=0}^{\ell} V_i$, such that $V_0 = \{v_0\}$, $V_\ell = T$, 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.*

Every branching program of size s can be efficiently transformed into a layered branching program of size at most s^2 and same length (cf. [32]). For convenience, we assume in our protocol that the server's BP is layered, which may square the server's work but has no effect on the communication complexity or the client's work. The quadratic overhead in the server's work can be avoided in most useful special cases (e.g., evaluating decision trees or finite automata) and can be avoided in the general case if only client privacy is required.

3 Cryptographic Primitives

In this section we define both our goal of computing on encrypted data and the main cryptographic tool on which we rely.

3.1 Computing on Encrypted Data

We consider a scenario where a client, holding an input x, publishes a public key pk and an encryption c of x under pk. This encryption is used by a server to efficiently evaluate a program P (in some given representation model) on c, obtaining a ciphertext c'. The client then uses its secret key to recover $P(x)$ from c'. This is formalized as follows.

Definition 5 (Computing on encrypted data). *Let $U : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$ be a representation model. A* protocol for evaluating programs from U on encrypted data *is defined by a tuple of algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *and proceeds as follows.*

- SETUP: *Given a security parameter k, the client computes $(pk, sk) \leftarrow \mathsf{Gen}(1^k)$ and saves sk for a later use.*
- ENCRYPTION: *The client computes $c \leftarrow \mathsf{Enc}(pk, x)$, where x is the input on which a program P should be evaluated.*
- EVALUATION: *Given the public key pk, the ciphertext c, and a program P, the server computes an encrypted output $c' \leftarrow \mathsf{Eval}(1^k, pk, c, P)$.*
- DECRYPTION: *Given the encrypted output c', the client outputs $y \leftarrow \mathsf{Dec}(sk, c')$.*

We require that if both parties act according to the above protocol, then for every input x, program P, and security parameter $k \in \mathbb{N}$, the output y of the final decryption phase is equal to $U(P, x)$ except, perhaps, with negligible probability in k.

An essential security requirement for computing on encrypted data is *client privacy*, requiring that the pair (pk, c) produced in the above process keep the client's input x semantically secure [17, 16].

Definition 6 (Client privacy). *Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ be a protocol for computing on encrypted data. We say that Π satisfies the* client privacy *requirement if the advantage of any PPT adversary* Adv *in the following game is negligible in the security parameter k:*

– Adv *is given 1^k and generates a pair $x_0, x_1 \in \{0,1\}^*$ such that $|x_0| = |x_1|$.*
– *Let $b \overset{\mathsf{R}}{\leftarrow} \{0,1\}$, $(pk, sk) \leftarrow \mathsf{Gen}(1^k)$, and $c \leftarrow \mathsf{Enc}(pk, x_b)$.*
– Adv *is given the challenge (pk, c) and outputs a guess b'.*

The advantage of Adv *is defined as $\mathbf{Pr}[b = b'] - 1/2$.*

Client privacy alone can be realized by simply letting Eval output P. However, it becomes nontrivial to satisfy when $|P| \gg |x|$ and the communication complexity is required to be sublinear in $|P|$. The latter requirement is in the center of this work.

While client privacy suffices for some applications, we will also be interested in protecting the privacy of the server by hiding the program P to the extent possible. For simplicity we consider here the case of a semi-honest client, who generates a valid public key pk and ciphertext c. The case of malicious clients will be addressed in Section 5.

Definition 7 (Server privacy: semi-honest model). *Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ be a protocol for evaluating programs from a representation model U on encrypted data. We say that Π has* statistical server privacy in the semi-honest model *if there exists a PPT algorithm* Sim *and a negligible function $\epsilon(\cdot)$ such that the following holds. For every security parameter k, input $x \in \{0,1\}^*$, pair (pk, c) that can be generated by* Gen, Eval *on inputs k, x, and program $P \in \{0,1\}^*$, we have*

$$\mathsf{SD}(\mathsf{Eval}(1^k, pk, c, P)\,,\, \mathsf{Sim}(1^k, 1^{|x|}, pk, U(P,x), 1^{|P|})) \leq \epsilon(k).$$

The case of perfect server privacy *is defined similarly, except that $\epsilon(k) = 0$ and* Sim *is allowed to run in expected polynomial time.*

In the case of computational server privacy*,* Sim *should satisfy the following requirement. For every polynomial-size circuit family D there is a negligible function $\epsilon(\cdot)$ such that for any k, x, pk, c, P as above we have*

$$\Pr[D(\mathsf{Eval}(1^k, pk, c, P)) = 1] - \Pr[D(\mathsf{Sim}(1^k, 1^{|x|}, pk, U(P,x), 1^{|P|})) = 1] \leq \epsilon(k).$$

Our main protocol will have perfect server privacy. In fact, it will additionally hide the size of the server's input P from the client. We refer to this property as *size hiding*. This implies, in particular, that the length of c' must be independent of the length of P.

Definition 8 (Size hiding server privacy: semi-honest model). *We say that Π has (perfect, statistical, or computational)* size hiding server privacy *in the semi-honest model if it satisfies the requirements of Definition 7 with the following difference:* Sim *does not get the length of P as an input.*

Remark 2. Protocols Π which satisfy our definitions of client privacy (Definition 6) and standard server privacy (Definition 7) can be easily derived from previous protocols for one-round secure computation. In particular, Yao's protocol [36] yields a protocol for evaluating circuits on encrypted data with computational server privacy, and protocols from [35, 21, 14, 18, 4, 22] yield protocols for evaluating formulas, branching programs, and even non-deterministic branching programs on encrypted data with perfect or statistical server privacy. However, in all these protocols the length of c' is generally bigger than the length of P. In particular, none of these protocols satisfies the additional size hiding property of Definition 8.

3.2 Oblivious Transfer

It will be convenient to present our main protocol in a modular way, using a variant of one-round Oblivious Transfer (OT) [33, 12] as a subprotocol. To this end it will be necessary to rely on a stronger server privacy property than the one implied by standard definitions of OT. As before, we focus here on the case of a semi-honest client and postpone the treatment of malicious clients to Section 5.

A standard one-round OT protocol 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 random input, the client should be able to recover s_i. The standard security requirements include *client privacy*, requiring that q keep i hidden from the server, and *server privacy,* requiring that a keep all secrets other than s_i hidden from the client. Note that the latter server privacy requirement does not rule out the possibility that a reveals information about the query q which is not implied by the output s_i alone. (In fact, a can include the entire query q without violating server privacy.) This might compromise the security of our main protocol, in which the client issues multiple OT queries and each query is used by the server to compute multiple answers. It will be crucial for the security of the protocol that the client be unable to correlate answers with queries, beyond correlations which follow from the outputs. Such correlations will reveal to the client information about the structure of the server's branching program.

Roughly speaking, our notion of strong OT strengthens the above server privacy requirement by requiring the distribution of the answer a conditioned on the output s_i to be independent of the query q. In other words, the distribution of the answer depends on the output alone. It turns out that a natural implementation of one-round OT based on homomorphic encryption [23, 34] satisfies the required properties (see Section 4.1). We now formally define strong OT.

Definition 9 (Strong OT). *A* strong OT *protocol is defined by a tuple of PPT algorithms* $(\mathsf{G}_{\mathsf{OT}}, \mathsf{Q}_{\mathsf{OT}}, \mathsf{A}_{\mathsf{OT}}, \mathsf{D}_{\mathsf{OT}})$. *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, τ are given as inputs to both parties. The protocol proceeds as follows:*

- *The client generates* $(pk, sk) \leftarrow \mathsf{G}_{\mathsf{OT}}(1^k)$*, computes a query* $q \leftarrow \mathsf{Q}_{\mathsf{OT}}(pk, 1^t, 1^\tau, i)$*, and sends* (pk, q) *to the server.*
- *The server computes* $a \leftarrow \mathsf{A}_{\mathsf{OT}}(pk, q, s_1, \ldots, s_t)$ *and sends a to the client.*
- *The client computes and outputs* $\mathsf{D}_{\mathsf{OT}}(sk, a)$*.*

We require that if both parties follow the protocol, the client always outputs s_i. We denote the length of the query q by $\alpha(k, t, \tau)$ and the length of the answer a by $\beta(k, t, \tau)$.

Our main protocol will require $\beta(k, t, \tau) = \tau + \text{poly}(k, t)$ to efficiently accommodate BPs of arbitrary length. (In fact, it suffices that the above holds for $t = 2$.) This will be our default efficiency requirement. However, this requirement can be relaxed if one settles for weaker forms of our main result that apply to shallow BPs, such as constant-length BPs over a polynomial-size input alphabet.

We now define the client privacy and (strong) server privacy requirements.

Definition 10 (Strong OT: client privacy). *We require that the client's query q keep i semantically secure. That is, the advantage of any PPT adversary* Adv *in the following game is negligible in the security parameter k:*

– Adv *is given 1^k and generates $1^t, 1^\tau$ and i_0, i_1 such that $i_0, i_1 \in [t]$.*
– *Let $b \stackrel{\mathsf{R}}{\leftarrow} \{0, 1\}$, $(pk, sk) \leftarrow \mathsf{G}_{\mathsf{OT}}(1^k)$, and $q \leftarrow \mathsf{Q}_{\mathsf{OT}}(pk, 1^t, 1^\tau, i_b)$.*
– Adv *is given the challenge (pk, q) and outputs a guess b' for b.*

The advantage of Adv *is defined as $\mathbf{Pr}[b = b'] - 1/2$.*

Our strong variant of perfect server privacy is defined similarly to Definition 7.

Definition 11 (Strong OT: server privacy). *There exists an expected polynomial time simulator* $\mathsf{Sim}_{\mathsf{OT}}$ *such that the following holds. For every k, t, τ, $i \in [t]$, pair (pk, q) that can be generated by $\mathsf{G}_{\mathsf{OT}}, \mathsf{Q}_{\mathsf{OT}}$ on inputs k, t, τ, i, and strings $s_0, \ldots, s_{t-1} \in \{0, 1\}^\tau$, the distributions $\mathsf{A}_{\mathsf{OT}}(pk, q, s_1, \ldots, s_t)$ and $\mathsf{Sim}_{\mathsf{OT}}(pk, 1^t, s_i)$ are identical.*

In the following it will sometimes be convenient to index the server's inputs s_i by $0, 1, \ldots, (t-1)$ instead of $1, 2, \ldots, t$.

4 Main Protocol

In this section we will describe our main protocol for evaluating branching programs on encrypted data. The protocol will provide client privacy, along with size hiding server privacy in the semi-honest model. Extensions that achieve server privacy in the malicious model will be presented in Section 5.

We fix a polynomially bounded length function $\ell(\cdot)$, and assume that if the client's input x is of length n, then the server's BP P is of length $\ell(n)$. (To conform to our general definition of representation models, one may define $P(x) = 0$ for P and x that do not match.) We also view the input domain I and output domain O as being implicitly determined by n. However, in the following it will be convenient to view ℓ, $|I|$, and $|O|$ as separate parameters which are given to both parties, and analyze the complexity of the protocol as a function of these parameters. We will also assume that P is layered (see Definition 4). As discussed in Section 2.1, every BP can be efficiently transformed into an equivalent layered BP without increasing its length.

Our protocol is based on a strong OT protocol as defined in Section 3.2 and proceeds roughly as follows. (For simplicity, assume that the input domain I of P is binary and that every nonterminal node in the graph has outdegree 2.) The client generates, for every input variable x_i and level j, an OT query q_i^j corresponding to a selection of the x_i-th string out of a pair of strings of an appropriate length. (This length will depend on j and will be later understood from the context.) The ℓn queries q_i^j jointly form the encryption c of x.

To evaluate P on c, the server makes a bottom-up pass on P, starting with the terminal nodes T and ending with the initial node v_0. This pass labels each node v in the graph by an OT answer which encrypts the output value to which x leads from this node. The pass consists of $\ell + 1$ iterations, where in iteration j ($0 \leq j \leq \ell$) all nodes of height j are handled. In iteration 0 every terminal node v is labeled by the corresponding output value $\psi_V(v)$. At the onset of the j-th iteration, $j \geq 1$, all nodes of height $j - 1$ have already been labeled. For each node v of height j, we want the labeling of v to encrypt the label of the child of v to which x leads. Such a label is computed by using the OT answering algorithm as follows. Suppose that the children of v are v_0 and v_1, where P branches from v to v_b if $x_i = b$. The label of v then computed by applying the OT answering algorithm to the query q_i^j on the pair of strings $(\mathsf{label}(v_0), \mathsf{label}(v_1))$. Note that since P is layered, the two labels have the same length. Moreover, by the strong server privacy property of the OT protocol, the label of v can be viewed as an encryption of the label of the selected child v_{x_i}. In particular, this label does not contain any information about the identity of the variable x_i that was used to determine the selection. (If a standard one-round OT is used, this is not necessarily be the case.)

Finally, at the end of iteration ℓ, the initial node v_0 is labeled by an OT answer which can be viewed as an (iterated) encryption of the output value $P(x)$. The client decrypts $P(x)$ by applying the OT decryption algorithm ℓ times to the label of v_0.

The above protocol is formally described in Figure 1. Its correctness is implied by the following lemma, which can be easily proved by induction on the height h.

Lemma 1. *For any node v, let $P_v(x)$ denote the output of P on the input x if v is used as the initial node. Then, for every $0 \leq h \leq \ell$ and every node v of height h we have ${\mathsf{D_{OT}}}^{(h)}(sk, \mathsf{label}(v)) = P_v(x)$, where ${\mathsf{D_{OT}}}^{(h)}(sk, \cdot)$ denotes the h-th iterate of $\mathsf{D_{OT}}(sk, \cdot)$.*

In particular, ${\mathsf{D_{OT}}}^{(\ell)}(sk, \mathsf{label}(v_0)) = P(x)$, from which correctness follows. We turn to analyze the protocol's efficiency.

Efficiency. Recall that we denote the length of an OT query by $\alpha(k, t, \tau)$ and the length of an OT answer by $\beta(k, t, \tau)$. Let β_j be as defined in Step 2, namely the result of applying the j-th iterate of $\beta(k, t, \cdot)$ on $\log |O|$. The length of the encryption c computed by the client is then bounded by $\ell n \cdot \alpha(k, t, \beta_\ell)$ and the length of the ciphertext c' computed by the server is $\beta_{\ell+1}$. By default, we assume the strong OT implementation to be such that $\beta(k, t, \tau) = \tau + \text{poly}(k, t)$. (See Section 4.1 for a concrete implementation using the Damgård-Jurik cryptosystem.) In such a case, the overall communication is $\text{poly}(k, n, \ell)$, which is in particular independent of $|P|$ as required. We will later present an optimized instantiation of the main protocol with a total communication of $O(kn\ell)$ (for the case of binary inputs and outputs). Finally, the computation performed by each party is polynomial in the length of its input.

Remark 3. When $\ell(n) \ll n$, the requirement that $\beta(k, t, \tau) = \tau + \text{poly}(k, t)$ can be relaxed. In particular, if $\ell(n) = O(\log n)$ it suffices that $\beta(k, t, \tau) = O(\tau) + \text{poly}(k, t)$.

Main Protocol

- Common inputs: security parameter 1^k, a branching program length parameter 1^ℓ, input domain $I = \{0, 1, \ldots, t-1\}$, output domain $O = \{0, 1\}^\gamma$.
- Client input: an assignment $x = (x_1, \ldots, x_n) \in I^n$.
- Server input: a layered BP $P = (G(V, E), v_0, T, \psi_V, \psi_E)$ of length ℓ.
- Sub-protocol: a strong OT protocol $(\mathsf{G_{OT}}, \mathsf{Q_{OT}}, \mathsf{A_{OT}}, \mathsf{D_{OT}})$ with answer length $\beta(k, t, \tau)$.

1. **Setup** $\mathsf{Gen}(1^k)$:
   - Let $(pk, sk) \leftarrow \mathsf{G_{OT}}(1^k)$.
   - Return (pk, sk).
2. **Encryption** $\mathsf{Enc}(pk, x)$:
   - For $1 \leq i \leq n$, generate a vector $q_i = (q_i^1, \ldots, q_i^\ell)$, where q_i^j is obtained by:
   $$q_i^j \leftarrow \mathsf{Q_{OT}}(pk, 1^t, 1^{\beta_j}, x_i),$$
   and where the lengths β_j are defined by $\beta_1 = \log |O|$ and $\beta_{j+1} = \beta(k, t, \beta_j)$.
   - Return $c = (q_1, \ldots, q_n)$.
3. **Evaluation** $\mathsf{Eval}(1^k, pk, c = (q_i^j), P)$:
   - Initialization: for each $v \in T$ set $\mathsf{label}(v) \longleftarrow \psi_V(v)$.
   - While v_0 isn't labeled:
     - Pick an unlabeled node $v \in V - T$ such that all its children are labeled.
     - Let $i \leftarrow \psi_V(v)$ and $h \leftarrow \mathsf{height}(v)$.
     - Let $\mathsf{label}(v) \leftarrow \mathsf{A_{OT}}(pk, q_i^h, \mathsf{label}(u_0), \ldots, \mathsf{label}(u_{t-1}))$, where u_m is the (unique) node such that $m \in \psi_E(v, u_m)$.
       Note that the nodes u_m are not necessarily distinct.
   - Return $c' = \mathsf{label}(v_0)$.
4. **Decryption** $\mathsf{Dec}(sk, c')$:
   - Let $d_\ell \leftarrow c'$.
   - For $j = \ell$ down to 1, let $d_{j-1} \leftarrow \mathsf{D_{OT}}(sk, d_j)$.
   - Return d_0.

Fig. 1. Evaluating a branching program on encrypted data

A strong OT protocol with the latter efficiency requirement can be based on homomorphic cryptosystems which expand the ciphertext length by a constant factor, such as El-Gamal (see Section 4.1). If $\ell(n) = O(1)$, we can rely on an arbitrary strong OT, which in turn can be based on an arbitrary homomorphic encryption scheme (including, for instance, the Goldwasser-Micali cryptosystem [17]).

Remark 4. The PIR protocol of [23] can be viewed as an instance of our construction in which ℓ is set to some constant d, the input domain I is of size $t = N^{1/d}$ (where N is the database size), and the database is represented as a complete decision tree of depth d and degree $N^{1/d}$. Its variant suggested in [34] (resp., [26]) corresponds to a decision tree of depth $\sqrt{\log N}$ and degree $t = 2^{\sqrt{\log N}}$ (resp., depth $\log N$ and degree $t = 2$). These three depth parameters correspond to the different BP length regimes discussed in Remark 3.

We turn to prove the security properties of the main protocol. In the following we assume that the given strong OT subprotocol is secure and that its answer complexity is $\beta(k, t, \tau) = \tau + \text{poly}(k, t)$. In Section 4.1 we will show that this assumption is implied by the DCRA assumption.

Theorem 1. *The protocol described in Figure 1 provides client privacy according to Definition 6 as well as perfect size hiding server privacy in the semi-honest model according to Definition 8.*

Proof sketch: Client privacy readily follows from the client privacy requirement in the underlying OT protocol. The security of sending polynomially many strong OT queries under the same key follows from the security of encrypting multiple messages under the same key in public-key encryption schemes (see [16], Theorem 5.2.11).

To prove size hiding server privacy, we describe a perfect simulator Sim. The idea is to recreate the labels of the computation path from v_0 to a terminal node labeled with $P(x)$ without knowing the nodes traversed by the path. Sim will use the OT simulator $\mathsf{Sim}_{\mathsf{OT}}$ as a subroutine. On inputs $(1^k, 1^{|x|}, pk, P(x))$ (and given $|I| = t$ as an additional public input), Sim proceeds as follows:

– Let $\ell \leftarrow \ell(|x|)$, $\lambda_0 \leftarrow P(x)$.
– For $j = 1$ to ℓ, let $\lambda_j \leftarrow \mathsf{Sim}_{\mathsf{OT}}(pk, 1^t, \lambda_{j-1})$.
– Return λ_ℓ.

Consider the computation path $v_0, v_1, \ldots, v_\ell$ induced by x. It follows by induction on j that the distribution of λ_j produced by Sim is identical to the distribution of $\mathsf{label}(v_{\ell-j})$ produced by $\mathsf{Eval}(1^k, pk, c, P)$, for every k, x, P and pair (pk, c) which can be generated by $\mathsf{Gen}, \mathsf{Enc}$ on k, x. In particular, the simulator's output λ_ℓ is distributed identically to $c' = \mathsf{label}(v_0)$. Note that the strong OT requirement allows $\mathsf{Sim}_{\mathsf{OT}}$ to produce the correct distributions independently of the OT queries included in c. □

4.1 Implementing Strong OT

Our concrete implementation of strong OT is based on the Damgård-Jurik (DJ) homomorphic public-key cryptosystem [11], which generalizes Paillier's cryptosystem [31].

It is suitable for our needs because it allows us to encrypt a group element of length τ into a ciphertext of length $\tau + O(k)$, where k is a security parameter. This efficiency feature is unique among all known homomorphic encryption schemes and is needed for our main protocol to be efficient for arbitrary length bounds $\ell(n)$. The semantic security of the DJ cryptosystem follows from the Decisional Composite Residuosity Assumption (DCRA) [11].

We now describe the main properties of the DJ cryptosystem that are useful for our purposes (see [11] for further details).

- KEY GENERATION: Given a security parameter k, $Gen(1^k)$ outputs a secret key (p_1, p_2), where p_1, p_2 are random k-bit primes (i.e., $2^{k-1} \geq p_1, p_2 < 2^k$), and a public key $N = p_1 p_2$. The above choice of p_1, p_2 guarantees that $gcd(N, \phi(N)) = 1$. This property will be useful in what follows. We refer to N which can be generated by $Gen(1^k)$ as a *valid DJ key*.
- ENCRYPTION: The DJ cryptosystem is length-flexible in the sense that every fixed key N allows to encrypt plaintexts of an arbitrary (polynomial) length, where the encryption only *adds* $O(k)$ bits to the length of the plaintext. Given a plaintext length parameter e, where $1 \leq e < \min(p_1, p_2)$, we define a plaintext group $M_{N,e} = \mathbb{Z}_{N^e}$ and a ciphertext group $C_{N,e} = \mathbb{Z}^*_{N^{e+1}}$. The restriction on e is required for correct decryption, and since we will only use $e \leq \text{poly}(k)$ it will always hold. Now fix some valid pair (N, e). To abbreviate notation we denote the ciphertext group $C_{N,e} = \mathbb{Z}^*_{N^{e+1}}$ by C. Let $C_0 = C^{N^e} = \left\{c^{N^e} | c \in C\right\}$. Clearly, C_0 is a subgroup of C. Let $g = N + 1 \in C$. The output distribution of the encryption is specified via an injective homomorphism $H : \mathcal{M}_{N,e} \rightarrow C/C_0$ defined by $H(m) = g^m \cdot C_0$, where $g^m \cdot C_0$ denotes the coset represented by g^m in C/C_0. To encrypt $m \in M_{N,e}$, the encryption function $E_{N,e}(m)$ returns a random element in the coset $H(m)$. This can be done by sampling $r \stackrel{\mathsf{R}}{\leftarrow} Z^*_N$ and outputting $c = g^m \cdot r^{N^e}$ (where all multiplications are in C). In particular, an encryption of 0 is a random element of C_0. Note that the difference between the size of the ciphertext ($\lceil \log N^{e+1} \rceil$) and the size of the plaintext ($\lceil \log N^e \rceil$) is indeed only $O(k)$.
- DECRYPTION: Given $c = g^m \cdot r^{N^e}$ and the factorization (p_1, p_2) of N, it is possible to efficiently decrypt m. We denote the decryption algorithm by $D_{(p_1,p_2),e}(c)$.
- HOMOMORPHISM: Given two ciphertexts $c \in E_{N,e}(m)$ and $c' \in E_{N,e}(m')$, their product $c \cdot c'$ (in the ciphertext group) is a valid encryption of the sum $m + m'$ (in the plaintext group). It follows that c^ρ is an encryption of $\rho \cdot m$. Moreover, multiplying c by a random encryption of 0 *rerandomizes* c into a random encryption of m.

Strong OT from the DJ cryptosystem. The following strong OT protocol is similar to the PIR protocol of [23] and its generalizations from [34, 26]. The choice of DJ as the underlying cryptosystem is motivated by the goal of handling branching programs of an arbitrary length. If the length function $\ell(n)$ is small, other homomorphic cryptosystems can be used (see Remark 3).

Construction 2 (Strong OT) Let $(Gen, E_{N,e}, D_{(p_1,p_2),e})$ be the DJ cryptosystem. The OT protocol $(\mathsf{G}_{\mathsf{OT}}, \mathsf{A}_{\mathsf{OT}}, \mathsf{Q}_{\mathsf{OT}}, \mathsf{D}_{\mathsf{OT}})$ proceeds as follows.

1. $\mathsf{G}_{\mathsf{OT}}(1^k)$:

- Let $(N,(p_1,p_2)) \leftarrow Gen(1^k)$.
- Return $(N,(p_1,p_2))$.

2. $\mathsf{Q_{OT}}(N, 1^k, 1^t, 1^\tau, i)$:
   - Let e be the minimal integer such that $N^e > 2^\tau$. We naturally identify strings in $\{0,1\}^\tau$ with integers in $M_{N,e} = Z_{N^e}$, and assume that elements in the groups $M_{N,e}$ and $C_{N,e}$ are padded so that their representation reveals e.
   - Let $q_i \leftarrow E_{N,e}(1)$ and $q_j \leftarrow E_{N,e}(0)$ for all $j \in [t] \setminus i$.
   - Return $q = (q_1, \ldots, q_{t-1})$.
3. $\mathsf{A_{OT}}(N, q, s_1, \ldots, s_t)$:
   - Infer e from q.
   - Let $q_t \leftarrow E_{N,e}(1) \cdot (\prod_{i=1}^{t-1} q_i^{s_i})^{-1}$ (where all operations are in $C_{N,e}$).
   - Let $a \leftarrow \prod_{i=1}^{t} q_i^{s_i} \cdot E_{N,e}(0)$.
   - Return a.
4. $\mathsf{D_{OT}}((p_1,p_2), a)$:
   - Infer e from a.
   - Return $D_{(p_1,p_2),e}(a)$.

Analysis. Correctness follows by observing that $(q_1, \ldots, q_t)$ encrypt the i-th unit vector of length t and a encrypts the inner product of $(s_1, \ldots, s_t)$ with this vector, which yields s_i. Client privacy follows from the semantic security of the DJ cryptosystem, which can be based on the DCRA assumption [11]. Server privacy follows from the fact that (due to rerandomization) the server's answer on any valid q is a *random* encryption of s_i, which can be easily generated by $\mathsf{Sim_{OT}}$. The protocol's query length is $\alpha(k,t,\tau) = t \cdot (\tau + O(k))$ and its answer length is $\beta(k,t,\tau) = \tau + O(k)$.

4.2 Optimizations

Optimizing the server's work. Our main protocol requires the branching program P to be layered. Converting an arbitrary BP to an equivalent layered BP of the same length may generally result in a quadratic blowup to its size, which in turn results in a quadratic computational overhead on the server's part. (We note, however, that most "natural" BPs, including ones that arise from other computation models such as finite automata, are either already layered or can be turned into equivalent layered BPs with only a linear overhead.) The quadratic overhead can be easily avoided in general if only client privacy is required. The main protocol can be modified in this case to operate on a non-layered BP by padding the labels that serve as OT inputs to match the size of the longest label.

Optimizing the encryption length. In the main protocol, the length of the encryption c produced by Enc must be bigger than $\sum_{j=1}^{\ell} \beta_j > \ell^2$. It turns out that the quadratic dependence on ℓ can be avoided by exploiting the specific structure of the DJ cryptosystem. The improvement is based on the following observation:

Observation 3 *For every valid DJ key pair (N,e), $e' < e$, $m \in M_{N,e}$ and $c \in E_{N,e}(m)$ (i.e., c is some valid encryption of m) it holds that*

$$c \bmod N^{e'+1} \in E_{N,e'}(m \bmod N^{e'}).$$

It follows from Observation 3 that the ciphertext c may consist of n encryptions q_i in the largest group (rather than n encryptions q_i^j for every level j of the BP), since the server can convert encryptions from the largest group into encryptions from smaller groups. (Note that since we only encrypt 0's and 1's, the conversion does not modify the encrypted value.) The improved implementation achieves communication complexity of $O(kn\ell)$ bits from the client to the server (instead of $O(kn\ell^2)$ in the original implementation) and $O(k\ell)$ bits from the server to the client (as in the original implementation). Clearly, the optimization doesn't compromise client or server privacy. Thus, we have:

Theorem 4. *Assuming DCRA [11], there is a protocol for evaluating a binary branching program of length ℓ and of arbitrary size on an encrypted input of length n, with a total communication of $O(kn\ell)$ bits (where k is a security parameter). The protocol provides client privacy as well as size hiding server privacy in the semi-honest model.*

5 Handling Malicious Clients

In this section we sketch the required modifications for achieving security against malicious clients. For lack of space we only describe the high level ideas and refer the reader to the full version for further details. For simplicity, we restrict the attention throughout this section to the case of branching programs over binary inputs.

We start by observing that a malicious client can easily break the server privacy of the main protocol even if it honestly generates the public key pk.

Example 1. Consider a client who sends an encryption of 2 (instead of 0 or 1) as an OT query. In this OT invocation, the client can recover both s_0 and s_1. This potentially reveals additional information about the structure of the branching program P. For instance, in the degenerate case where P consists of an initial node and two terminal nodes, the client will learn the values of both terminal nodes.

The above mild form of cheating is relatively easy to handle using previous techniques [15, 1, 24] and will be addressed in Section 5.1. A more challenging goal is to handle clients that are also free to choose invalid public keys pk. This scenario will be addressed in Section 5.2.

Before describing our solutions, we formalize our notions of server privacy in the malicious model. The following definitions modify Definition 8 in that they allow an unbounded simulator to extract an effective input x^* from a corrupted ciphertext c^* and a (possibly) corrupted public key pk^*. The use of unbounded simulation seems necessary in the "vanilla" one-round malicious model (i.e., without setup assumptions) and was previously made in similar contexts [29, 1, 13, 20, 24].

We start by defining the *trusted setup model*, where the client is forced to use a valid public key pk but can cheat by creating invalid ciphertexts c^*. This model is motivated by the fact that the same public key may be reused to encrypt many different inputs. Thus, one can afford an expensive certification procedure (e.g., using interactive zero-knowledge proofs or a trusted party) that is used once and for all.

Definition 12 (Size hiding server privacy: trusted setup model). *Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ be a protocol for evaluating programs from a representation model U on encrypted data. We say that Π has* statistical server privacy in the trusted setup model *if there exists a computationally unbounded, randomized algorithm* Sim *and a negligible function $\epsilon(\cdot)$ such that the following holds. For every security parameter k, valid public key pk that can be generated by $\mathsf{Gen}(1^k)$, and arbitrary ciphertext c^* there exists an "effective" input x^* such that for every program $P \in \{0,1\}^*$, we have*

$$\mathsf{SD}(\mathsf{Eval}(1^k, pk, c, P)\,,\, \mathsf{Sim}(1^k, pk, c^*, U(P, x^*))) \le \epsilon(k).$$

The case of computational *server privacy is defined in an analogous way (see Definition 7), where statistical indistinguishability is replaced by computational one.*

We turn to the fully malicious model.

Definition 13 (Size hiding server privacy: fully malicious model). *We say that Π has (statistical or computational)* server privacy in the fully malicious model *if it satisfies Definition 12 with the following modification: instead of quantifying over all* valid *public keys pk, now the quantification is over arbitrary public keys pk^*.*

Protocols for computing on encrypted data in the above model give rise to one-round (two-message) protocols for secure two-party computation of $U(\cdot,\cdot)$ under the relaxed security definitions of [29, 1, 13].

A natural approach for handling malicious clients would be to leave the main protocol as it is and only upgrade the original strong OT primitive into one that achieves security against malicious clients. Unfortunately, we cannot use this modular approach for several reasons. First, the basic variant of the protocol requires the client to use each input x_i in multiple OT invocations (corresponding to the different levels where x_i appears) and so the client could cheat by simply using inconsistent inputs in these OT invocations. More importantly, we do not know how to construct a strong OT protocol which simultaneously satisfies both our security and efficiency requirements in the malicious model. It is interesting to note that a one-round OT protocol of Kalai [20], which is based on Paillier's cryptosystem and can be generalized to work with the DJ cryptosystem, fails with respect to both security (in that it is not a *strong* OT) and efficiency (in that its answer significantly blows up the length of the selected string). Still, ideas from [20] will be instrumental in our solution for the fully malicious model.

5.1 Trusted Setup Model

We now describe a solution in the trusted setup model. Our starting point is the optimized instantiation of the protocol for the semi-honest model (Section 4.2), where in the case of binary inputs ($t = 2$) the client sends a single encryption for each input. Our goal is to prevent the type of attack described in Example 1, namely to ensure that each encryption sent by the client is indeed an encryption of 0 or 1. To this end one could employ general-purpose zero-knowledge proofs, forcing the client to prove that its queries are well formed. However, this approach requires multiple rounds of interaction which we would like to avoid, and also involves a considerable efficiency overhead.

Instead, we apply the conditional disclosure of secrets (CDS) methodology of [15, 1]. The idea is that instead of making the client prove that its queries are well formed, it suffices for the server to disclose its answer c' to the client only under the condition that the queries are well formed. Using the homomorphic property of the encryptions, the latter conditional disclosure can be done without the server even knowing whether the condition is satisfied.

The original CDS solutions from [1] relies on homomorphic encryption over groups of a prime order. An efficient extension to groups of a composite order was suggested in [24], assuming that the order of the group is sufficiently "rough". We employ a similar extension which avoids the roughness assumption and is geared towards the solution in the fully malicious model.

We start by describing the approach of [1]. The simplest setting involves a server holding a (valid) public key pk of a homomorphic cryptosystem, a ciphertext $c \in E_{pk}(m)$ (presumably generated by a client), and a secret s. The client holds the secret key sk corresponding to pk. The goal is for the server to send a single (randomized) ciphertext $\tilde{c}$ such that: (1) if $m = 0$ then s can be recovered from $\tilde{c}$ using the secret key; and (2) if $m \neq 0$ then $\tilde{c}$ reveals (almost) no information about s. The above is referred to as a CDS of the secret s under the condition $m = 0$. A solution to this simple CDS problem can be easily extended to CDS under more general conditions, involving multiple inputs m_i and general predicates over atomic conditions of the form $m_i = b_i$. In particular, $2n$ invocations of the above primitive are sufficient to disclose a secret under the suitable condition here, namely that n ciphertexts c_i *all* encrypt 0/1 values.

The solution of [1] is to let $\tilde{c}$ be a random encryption of $s+\rho m$, where ρ is a random integer between 1 and the order of the plaintext group. Note that $\tilde{c}$ can be efficiently computed using the homomorphic properties of the encryption. Requirement (1) holds because if $m = 0$ then $\tilde{c}$ encrypts $s + \rho \cdot 0 = s$. Requirement (2) holds in the case where the plaintext group is of a prime order; indeed, in that case if $m \neq 0$ then ρm is uniformly distributed over the plaintext group and therefore can be used to hide s.

The next observation is that in the case that the plaintext group has a composite order, not all is lost. In this case, if $m \neq 0$ then ρm is uniformly distributed over a nontrivial subgroup of the plaintext group. If s is chosen uniformly at random from the plaintext group, then s will still have at least one bit of remaining entropy even when conditioned on $\tilde{c}$. This residual randomness can be extracted using standard privacy amplification techniques. Specifically, to disclose an l-bit secret we first repeat the above $l + k$ times with independent secrets s_i, increasing the conditional entropy to $l + k$, and then apply an arbitrary strong randomness extractor (e.g., a pairwise independent hash function) to extract l (almost) perfectly secret bits from the partially leaked secrets.

The above approach (or the similar approach from [24]) solves our problem in the trusted setup model. In this case, every possible string c^* can be interpreted as a valid ciphertext encrypting some message m in the plaintext group Z_{N^e}. Thus, we can use the above to disclose the server's answer under the condition that the n encryptions produced by the client are well formed. This yields a protocol for the trusted setup model whose communication complexity is comparable to that of the optimized version of the original protocol.[4]

5.2 Fully Malicious Model

The previous solution relied on the fact that for a valid key N, every c^* can be interpreted as a valid encryption of some message m. This does not hold in general. In fact, there is an explicit cheating strategy which uses N such that $\gcd(N, \phi(N)) > 1$ (e.g., $N = p_1 p_2$, where p_1, p_2 are odd primes and $p_2 = 2p_1 + 1$) in order to break the previous protocol. The main difficulty arises from the fact that the set of harmful keys N cannot be efficiently recognized. Our high level approach for getting around this problem is to project ciphertexts sent by the client onto a "harmless" subgroup of C by having the server raise them to the power of N^T, where $T = \lceil \log N \rceil$. To maintain correctness, plaintexts are chosen from a subgroup of Z_{N^e} of size N^{e-T}, which requires to moderately increase the values of e used in our protocol. We refer the reader to the full version for further details.

Acknowledgements. We thank Mike Freedman, Benny Pinkas, and Omer Reingold for discussions about secure keyword search from which this work originated. We also thank the anonymous TCC referees for many helpful comments and suggestions.

References

1. W. Aiello, Y. Ishai and O. Reingold. Priced oblivious transfer: How to sell digital goods. In *Proc. of EUROCRYPT 2001*, pages 119-135.
2. M. Blum, A. De Santis, S. Micali, and G. Persiano. Non-interactive Zero Knowledge. *SIAM Journal of Computing*, 20(6), pages 1084-1118, 1991.
3. B. Barak and O. Goldreich. Universal Arguments and their Applications. In *Proc. CCC 2002*, pages 194-203.
4. D. Beaver. Minimal-Latency Secure Function Evaluation. In *Proc. of EUROCRYPT 2000*, pages 335-350.
5. D. Boneh, E.J. Goh, and K. Nissim. Evaluating 2-DNF formulas on ciphertexts. In *Proc. 2nd TCC*, pages 325–341, 2005.
6. C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. *Proc. of EUROCRYPT '99*, pages 402–414.
7. C. Cachin, J. Camenisch, J. Kilian, and J. Muller. One-round secure computation and secure autonomous mobile agents. In *Proceedings of ICALP '00*.
8. R. Canneti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1), pages 143-202.
9. B. Chor, N. Gilboa, and M. Naor. Private information retrieval by keywords. Technical Report TR-CS0917, Department of Computer Science, Technion, 1997.
10. B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. *J. of the ACM*, 45:965–981, 1998. Earlier version in FOCS '95.

[4] This holds for the case of *computational* server privacy, where we can afford to disclose a short secret s and then encrypt the (long) answer using this key. The statistically private variant involves an additional multiplicative overhead of $O(\ell)$.

11. I. Damgård and M. Jurik. A Generalisation, a Simplification and some Applications of Paillier's Probabilistic Public-Key System. In *Proc. of CT-RSA '02*, pages 79-95.
12. S. Even, O. Goldreich and A. Lempel. A Randomized Protocol for Signing Contracts. In *Communications of the ACM,* 28(6):637–647, 1985.
13. M.J. Freedman, Y. Ishai, B. Pinkas and O. Reingold. Keyword search and oblivious pseudorandom fuctions. In *Proc. of TCC 2005* vol. 3378, pages 303-324.
14. U. Feige, J. Kilian, and M. Naor. A minimal model for secure computation. In *Proc. of 26th STOC*, pages 554-563, 1994.
15. Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting Data Privacy in Private Information Retrieval Schemes. In *Proc. of 30th STOC*, pages 151-160, 1998.
16. O. Goldreich. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, 2004.
17. S. Goldwasser and S. Micali. Probabilistic encryption. *JCSS*, 28(2):270–299, 1984. Preliminary version in Proc. STOC '82.
18. Y. Ishai and E. Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *Proc. 41st FOCS*, pp. 294–304, 2000.
19. Y. Ishai and E. Kushilevitz. Perfect Constant-Round Secure Computation via Perfect Randomizing Polynomials. In *Proc. of the 29th ICALP*, pages 244-256, 2002.
20. Y. T. Kalai. Smooth Projective Hashing, and two message Oblivious Transfer. In *Proc. of EUROCRYPT 2005*, pages 78-95.
21. J. Kilian. Founding cryptography on oblivious transfer. In *Proc. of the 20th ACM*, pages 20-31, 1998.
22. V. Kolesnikov. Gate Evaluation Secret Sharing and Secure One-Round Two-Party Computation. In *Proc. of ASIACRYPT 2005*, pages 136-155.
23. E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval In *Proc. 38th FOCS*, pages 364-273, 1997.
24. S. Laur and H. Lipmaa. Additively homomorphic Conditional Disclosure of Secrets and applications. Eprint report 2005/378.
25. Y. Lindell and B. Pinkas. A Proof of Yao's Protocol for Secure Two-Party Computation. Cryptology ePrint Archive, Report 2004/175, 2004.
26. H. Lipmaa. An Oblivious Transfer Protocol with Log-Squared Communication. In *Proc. 8th ICS*, pages 314-328, 2005. Full version on eprint.
27. S. Micali, M. Rabin and J. Kilian. Zero knowledge sets. In *Proc. 44th FOCS*, pages 80-91, 2003.
28. M. Naor and K. Nissim. Communication Preserving Protocols for Secure Function Evaluation. In *Proc. 33rd STOC*, pages 590–599, 2001.
29. M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In Proc. SODA 2001.
30. R. Ostrovsky and W. E. Skeith III. Private Searching on Streaming Data. In *Proc. Crypto 2005*, pages 223-240.
31. P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Proc. of EUROCRYPT 1999*, pages 223-238.
32. N. Pippenger. On simultaneous resource bounds. In *Proc. of the 20th FOCS*, pages 307-311, 1979.
33. M. Rabin. How to Exchange Secrets by Oblivious Transfer. Tech. Memo TR-81, Aiken Computation Laboratory, Harvard U., 1981.
34. J.P. Stern. A new and efficient all or nothing Disclosure of Secrets protocol. In *Proc. AsiaCrypt 98*, vol. 1514, pages 357-371.
35. T. Sander, A. Young and M. Yung. Non-interactive cryptocomputing for NC_1. In *Proc. 20th FOCS*, pages 554-566, 1999.
36. A.C. Yao. How to generate and exchange secrets. In *Proc. 18th STOC*, pages 162-167, 1986.