Secure Function Evaluation with Ordered Binary Decision Diagrams

Louis Kruger and Somesh Jha
University of Wisconsin-Madison
[lpkruger,jha]@cs.wisc.edu

Eu-Jin Goh and Dan Boneh
Stanford University
[eujin, dabo]@cs.stanford.edu

ABSTRACT

Privacy-preserving protocols allow multiple parties with private inputs to perform joint computation while preserving the privacy of their respective inputs. An important cryptographic primitive for designing privacy-preserving protocols is secure function evaluation (SFE). The classic solution for SFE by Yao uses a gate representation of the function that the two parties want to jointly compute. Fairplay is a system that implements the classic solution for SFE. In this paper, we present a new protocol for SFE that uses a graph-based representation of the function. Specifically we use the graph-based representation called ordered binary decision diagrams (OBDDs). For a large number of Boolean functions, OBDDs are more succinct than the gate-based representation. Preliminary experimental results based on a prototype implementation shows that for several functions, our protocol results in a smaller bandwidth than Fairplay. For example, for the classic millionaire's problem, our new protocol results in a approximately 45% bandwidth reduction over Fairplay. Therefore, our protocols will be particularly useful for applications for environments with limited bandwidth, such as applications for wireless and sensor networks.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: Security and Protection

General Terms

Security, Algorithms, Theory

Keywords

Binary Decision Diagrams, Secure Function Evaluation

1. INTRODUCTION

The ease and transparency of information flow on the Internet has heightened concerns of personal privacy [10, 25]. Various Internet activities, such as Web surfing, email, and other services leak sensitive information. As a result, there has been interest in developing technologies [9, 13, 24] and protocols to address these concerns. In particular, privacy-preserving protocols [11, 12, 18, 20] that allow multiple parties to perform joint computations without revealing their private inputs have been the subject of much interest. Our focus in this paper is on two party privacy-preserving protocols.

One of the fundamental cryptographic primitives for designing privacy-preserving protocols is *secure function evaluation (SFE)*. A protocol for SFE enables two parties A and B with inputs x and y respectively to jointly compute a function $f(x, y)$ while preserving the privacy of the two parties (i.e., at the end of the protocol, party A only knows its input x and the value of the function $f(x, y)$, and similarly for B). Yao showed that for a polynomial-time computable function f, there exists a SFE protocol that executes in polynomial time [15, 27] (details about this protocol can be found in Goldreich's book [14, Chapter 7]). Yao's classic solution for SFE has been used to design privacy-preserving protocols for various applications [1]. The importance of Yao's protocol spurred researchers to design a compiler that takes a description of the function f and emits code corresponding to Yao's protocol for secure evaluation of f. Such compilers, for example Fairplay [22], enable wider applicability of SFE. MacKenzie *et al.* [21] implemented a compiler for generating secure two-party protocols for a restricted but important class of functions, which is particularly suited for applications where the secret key is protected using threshold cryptography. For most applications, the classic protocol for SFE is quite expensive, which has led researchers to develop more efficient privacy-preserving protocols for specific problems [11, 12, 18, 20].

In the classic SFE protocol, the function f is represented as circuit comprised of gates. Fairplay uses this circuit representation of f. *Ordered Binary Decision Diagrams (OBDDS)* are a graph-based representation of Boolean functions that have been used in a variety of applications in computer-aided design, including symbolic model checking (a technique for verifying designs), verification of combinational logic, and verification of finite-state concurrent systems [3, 7]. OBDDs can be readily extended to represent functions with arbitrary domains and ranges.

Given an OBDD representation of the function to be jointly computed by the two parties, Yao's protocol can be directly used by first converting the OBDD into a circuit. Converting an OBDD to a circuit, however, incurs a blow-up in

CCS'06, October 30–November 3, 2006, Alexandria, Virginia, USA.

the number of gates required. To empirically measure this blowup, we implemented a compiler that takes an OBDD and converts it into a circuit description that can be used in Fairplay. On the average, this conversion from OBDD to circuit resulted in a increase in size by a factor of 10. Details of this experiment can be found in Section 4.

In this paper, we present a SFE algorithm that directly uses an OBDD representation of the function f that the two parties want to jointly compute. The advantage of using an OBDD representation over the gate-representation is that OBDDs are more succinct for certain widely used classes of functions than the gate representation. For example, among other functions, our results show the OBDD representation is more efficient than the gate representation for 8-bit AND, 8-bit addition, and the millionaire's and billionaire's problems [27]. As a result, our protocol has reduced bandwidth consumption over the classic Yao protocol implemented in Fairplay. Because processor speeds have increased at a more rapid pace than bandwidth availability over the past years, network bandwidth is likely to be the bottleneck for a number of applications. In particular, our protocols are especially useful for applications operating over networks with limited bandwidth, such as wireless and sensor networks. Furthermore, we have empirically confirmed this statement by implementing our protocol and comparing it with Fairplay.

This paper makes the following contributions:

- We present a SFE protocol that uses the OBDD representation of the function to be jointly computed by two parties. Our new protocol along with the correctness proof is provided in Section 3.
- Experimental results based upon a prototype implementation of our protocol demonstrate that for certain functions, our implementation results in a smaller encrypted circuit than Fairplay. For example, for the classic millionaire's problem, our implementation reduces the bandwidth by approximately 45% over Fairplay. Our implementation and experimental results are described in Section 4.

In summary, this paper presents a new SFE protocol that uses the OBDD representation. The OBDD representation is more efficient for several practical functions of interest. For other functions, the circuit description (and therefore FairPlay) will be more efficient. This paper presents a generic alternative to Boolean circuits that can be used when appropriate.

2. ORDERED BINARY DECISION DIAGRAMS (OBDDS)

Ordered binary decision diagrams (OBDDs) are a canonical representation for Boolean formulas [3]. They are often substantially more compact than traditional normal forms, such as conjunctive normal form (CNF) and disjunctive normal form (DNF), and they can be manipulated efficiently. Therefore, they are widely used for a variety of applications in computer-aided design, including symbolic model checking, verification of combinational logic, and verification of finite-state concurrent systems [7]. A detailed discussion of OBDDs can be found in Bryant's seminal article [3].

Given a Boolean function $f(x_1, x_2, \cdots, x_n)$ of n variables $x_1, \cdots, x_n$ and a total ordering on the n variables, the OBDD for f, denoted by $OBDD(f)$, is a rooted, directed acyclic graph (DAG) with two types of vertices: *terminal* and *nonterminal* vertices. $OBDD(f)$ also has the following components:

- Each vertex v has a level, denoted by $level(v)$, between 0 and n. There is a distinguished vertex called *root* whose level is 0.
- Each nonterminal vertex v is labeled by a variable $var(v) \in \{x_1, \cdots, x_n\}$ and has two successors, $low(v)$ and $high(v)$. Each terminal vertex is labeled with either 0 or 1. There are only two terminal vertices in an OBDD. Moreover, the labeling of vertices respects the total ordering $<$ on the variables, i.e., if u has a nonterminal successor v, then $var(u) < var(v)$.

Given an assignment $\mathcal{A} = \langle x_1 \leftarrow b_1, \cdots, x_n \leftarrow b_n \rangle$ to the variables $x_1, \cdots, x_n$ the value of the Boolean function $f(b_1, \cdots, b_n)$ can be found by starting at the root and following the path where the edges on the path are labeled with $b_1, \cdots, b_n$. OBDDs can also be used to represent functions with finite range and domain. Let g be a function of n Boolean variables with output that can be encoded by k Boolean variables. The function g can be represented as an array of k OBDDs where the i-th OBDD represents the Boolean function corresponding to the i-th output bit of g. For the rest of the paper we will assume that the function f is a Boolean function, but our protocols can be easily extended for the case of functions with a finite range. We will illustrate OBDDs with an example.

EXAMPLE 1. *Figure 1 shows the OBDD for the function $f(x_1, x_2, x_3, x_4) = (x_1 = x_2) \wedge (x_3 = x_4)$ of four variables x_1, x_2, x_3, x_4 with the total ordering $x_1 < x_2 < x_3 < x_4$.*[1] *Notice that the ordering of the labels on the vertices on any path from the root to the terminals of the OBDD corresponds to the total ordering of the Boolean variables. Consider the assignment $\langle x_1 \leftarrow 1, x_2 \leftarrow 1, x_3 \leftarrow 0, x_4 \leftarrow 0 \rangle$. In the OBDD shown in Figure 1, if we start at the root and follow the edges corresponding to the assignment, we end up at the terminal vertex labeled with* 1. *Therefore, the value of $f(1, 1, 0, 0)$ is* 1.

One of the advantages of OBDDs is that they can be manipulated efficiently, i.e., given OBDDs for f and g, OBDDs for $f \wedge g$, $f \vee g$, and $\neg f$ can be computed efficiently. We now describe an operation called *restriction*, which is used in our protocol. Given a n variable Boolean function $f(x_1, x_2, \cdots, x_n)$ and a Boolean value b, $f\mid_{x_i \leftarrow b}$ is a Boolean function of $n-1$ variables $x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n$ defined as follows:
$f\mid_{x_i \leftarrow b}(x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n)$ is equal to
$f(x_1, \cdots, x_{i-1}, b, x_{i+1}, \cdots, x_n)$. Essentially $f\mid_{x_i \leftarrow b}$ is the function obtained by substituting the value b for the variable x_i in the function f. Given the OBDD for f, the $OBDD$ for $f\mid_{x_i \leftarrow b}$ can be efficiently computed [3, Section 4]. The restriction operation can be extended to multiple variables in a straightforward manner, e.g., $f\mid_{x_i \leftarrow b, x_j \leftarrow b'}$ can be computed as $(f\mid_{x_i \leftarrow b})\mid_{x_j \leftarrow b'}$. We explain the algorithm using our example; the reader is referred to [3] for details. Consider the function $f(x_1, x_2, x_3, x_4)$ described in example 1.

[1]OBDDs are sensitive to variable ordering, e.g., with the ordering $x_1 < x_3 < x_2 < x_4$ the OBDD for $(x_1 = x_2) \wedge (x_3 = x_4)$ has 11 nodes.

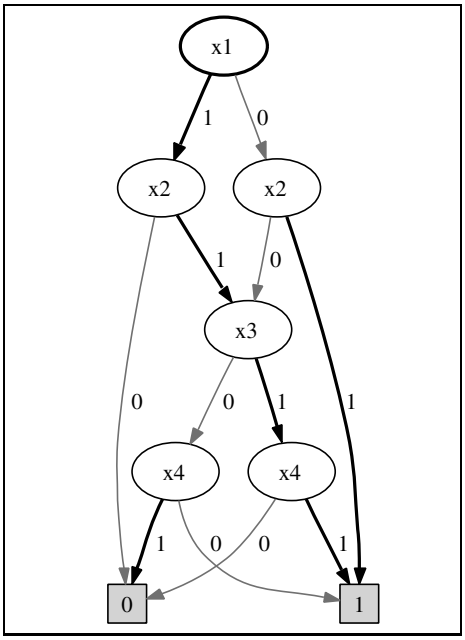

Figure 1: OBDD for the function $f(x_1, x_2, x_3, x_4) = (x_1 = x_2) \wedge (x_3 = x_4)$.

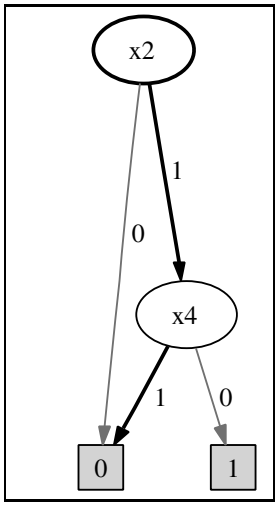

Figure 2: OBDD for the restriction $f\,|_{x_1 \leftarrow 1, x_3 \leftarrow 0}$ where $f(x_1, x_2, x_3, x_4) = (x_1 = x_2) \wedge (x_3 = x_4)$.

The OBDD corresponding to $f\,|_{x_1 \leftarrow 1, x_3 \leftarrow 0}$ is shown in Figure 2. Since $x_1 \leftarrow 1$, the root of OBDD ($f\,|_{x_1 \leftarrow 1, x_3 \leftarrow 0}$) is the left vertex labeled with x_2. Consider the two vertices v_1 and v_2 labeled with x_2. If v_1 has an edge that points to the vertex labeled with x_3, then that edge is changed to point to the right vertex labeled with x_4 (because this is the vertex reached if x_3 is equal to 1). Notice that in the reduced OBDD shown in Figure 2 the vertices that are labeled with x_1 and x_3 have been eliminated.

3. TWO PARTY SFE WITH OBDDS

For our protocols, we require a symmetric encryption scheme with two easily attained special properties [19], which are (1) *elusive range*: an encryption under one key is in the range of an encryption with a different key with negligible probability, and (2) *efficiently verifiable range*: given a key, a user can efficiently verify that a ciphertext is in the range of that key. These properties are required so that the receiver of the garbled OBDD can correctly decrypt nodes in the OBDD. The formal definition of these properties by Lindell and Pinkas [19] is provided with the proofs. An example of a symmetric key encryption scheme that fulfills these properties is $E_k(m) = (r\,,\, f_k(r) \oplus m \,\|\, 0^n)$, where $f : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^{2n}$ is a pseudo-random function and $r \stackrel{R}{\leftarrow} \{0,1\}^n$ is a n-bit random sequence. Unless stated otherwise, all symmetric key encryption schemes in this paper, besides being semantically secure [14, Chapter 5], also require these two properties.

Our protocol also uses a 1-out-of-2 oblivious transfer (denoted OT_1^2) protocol. A 1-out-of-k oblivious transfer OT_1^k is a protocol that lets Bob obtain one of k secrets held by Alice, without Alice learning which secret Bob obtains.

We now give the protocol for securely computing an OBDD between two parties where each party holds a part of the input. Assume f is a Boolean function $f(x_1, x_2, \cdots, x_n)$ of n Boolean variables $x_1, x_2, \cdots, x_n$. Let $OBDD(f)$ denote the OBDD for f with the ordering $x_1 < x_2 < \cdots < x_n$. We describe the protocol in stages. Protocol 1 described in Section 3.1 assumes that Alice holds inputs corresponding to the first k variables, and Bob has the inputs corresponding to last $n-k$ variables $x_{k+1}, \ldots, x_n$. Protocol 2 described in Section 3.2 allows arbitrary sharing of inputs, and it uses the restriction operation on OBDDs described earlier to reduce the bandwidth requirement of the protocol.

3.1 Protocol 1

For this protocol, we assume that Alice holds the inputs $(i_1, \ldots, i_k)$ corresponding to the first k variables $x_1, \ldots, x_k$, and Bob has the inputs $(i_{k+1}, \ldots, i_n)$ corresponding to last $n-k$ variables $x_{k+1}, \ldots, x_n$. In our protocol, Alice and Bob want to compute $f(x_1, \cdots, x_n)$ on their inputs using the OBDD for f. As the outcome, Bob learns $f(i_1, \ldots, i_n)$. This protocol is described in Figure 3.

One of the difficulties in developing and proving the protocol is that OBDDs allow skipping of levels. For example, Figure 4(a) shows the OBDD for the Boolean function $x_1 \wedge x_2$. Assume that the vertex at level 0 is labeled with x_1 and vertices at level 1 are labeled with x_2. Suppose Alice owns variable x_1 and Bob owns variable x_2. If Alice's input is 1, then Bob follows one more edge than if Alice's input were 0, which allows Bob to determine Alice's input. Compare this to Figure 4(b), where a dummy vertex is added so that, regardless of Alice's input, Bob has to follow the same number of edges. In this case, Bob learns nothing about Alice's input. Before Alice garbles the OBDDs, she adds dummy vertices so that each path from the root to a terminal node has the same number of edges. Alice adds dummy nodes whenever $OBDD(f)$ allows Bob to skip levels when evaluating the OBDD on his share of the input. Recall that the 0-successor and 1-successor of v is denoted by $low(v)$

Input: Both parties' inputs include the $OBDD(f)$ for the Boolean function $f(x_1, x_2, \cdots, x_n)$ with the ordering $x_1 < x_2 < \cdots < x_n$. Furthermore, Alice holds the inputs $(i_1, \ldots, i_k)$ corresponding to the first k variables $x_1, \ldots, x_k$, and Bob has the inputs $(i_{k+1}, \ldots, i_n)$.

1. Alice performs the following steps:
   (a) She traverses the $OBDD(f)$ using her input $(i_1, \cdots, i_k)$, which results in a node v_{init} at level k.
   (b) She uniformly and independently at random creates $(n-k)$ pairs of secrets $(s_1^0, s_1^1), \cdots, (s_{n-k}^0, s_{n-k}^1)$. In addition, for each node v in the $OBDD(f)$ whose level is between k and $n-1$, Alice also creates a secret s_v.
   (c) She assigns a uniformly random label to each node whose level is between k and n. We refer to the randomly assigned label of node v using the notation $label(v)$.
   (d) Next, Alice augments $OBDD(f)$ with some number of dummy nodes (to ensure that Bob always traverses $n-k$ nodes in his phase of the protocol).
   (e) Alice garbles all nodes whose level is between k and $n-1$ in the following manner. Let v be a node in $OBDD(f)$ such $k \leq level(v) \leq n-1$ and define $level(v) = \ell$. The encryption of node v, denoted by $E^{(v)}$, is a label and a randomly ordered ciphertext pair

$$\Big(label(v)\ ,\ E_{s_v \oplus s^0_{\ell-k+1}}(label(low(v)) \,\|\, s_{low(v)})\ ,\ E_{s_v \oplus s^1_{\ell-k+1}}(label(high(v)) \,\|\, s_{high(v)})\Big)\ ,$$

   where the labels are pre-pended to the secret with a separator symbol and the order of the ciphertexts is determined by a fair coin flip. Roughly speaking, the secrets corresponding to the 0-successor and 1-successor of node v are encrypted with the secret corresponding to v and its level.
   Note that dummy nodes have the same structure as normal nodes, except that the ciphertext pair contain encryptions of the same message since dummy nodes have the same 0 and 1-successors. Provided the encryption scheme is semantically secure, this poses no problem since the keys are chosen uniformly at random.
   Lastly, there are two terminal nodes of the form $(b, label(t_b))$ for $b = 0$ or 1. Recall that $OBDD(f)$ has two terminal nodes, denoted as 0 and 1, that are at level n.
   (f) Once Alice is done encrypting, she sends to Bob the encryption of all nodes whose level is between k and n and the secret $s_{v_{init}}$ corresponding to node v_{init} at level k. We called this the garbled OBDD.
2. Bob performs the following steps:
   (a) He engages in $n-k$ 1-out-of-2 oblivious transfers to obtain the secrets corresponding to his input. For example, if his input i_j is 0, then he obtains the (level) secret s_{j-k}^0; otherwise, he obtains the secret s_{j-k}^1.
   (b) Now Bob is ready to start his computation. Suppose $i_{k+1} = 0$. With s_1^0 and $s_{v_{init}}$, he decrypts both ciphertexts in $E^{(v_{init})}$ and decides which gives the correct result by using the verifiable range property of the encryption scheme. Bob now has both $s_{low(v)}$ (the secret corresponding to the 0-successor of v_{init}) and $label(low(v))$ (which tells Bob which encrypted node is used to evaluate his next input). Continuing this way, Bob eventually obtains a label corresponding to one of the terminal nodes, which determines the result of the OBDD on the shared inputs. Bob sends this result to Alice.

Figure 3: Protocol 1.

and $high(v)$. For example, if node n at level j has node n''' at level $j+3$ as its 0-successor, then Alice inserts two dummy nodes n' and n'' at level $j+1$ and $j+2$ respectively. The 0-successor of node n is changed to n', both 0 and 1-successors of n' are set to n'', and both 0 and 1-successors of n'' are set to n'''.

We prove correctness and security in the case of semi-honest parties. In the semi-honest model, both parties are assumed to perform computations and send messages according to their prescribed actions in the protocol. They may also record whatever they see during the protocol (i.e. their own input and randomness, and the messages they receive). We refer readers to Goldreich's book [14] for the complete definitions. Claim 1 proves that the protocol shown in Figure 3 is correct; that is, Bob computes the function $f(i_1, \cdots, i_n)$. Claim 2 proves that protocol 1 is secure in the semi-honest model; that is, at the end of protocol 1, Alice only knows its input and the value of the function $f(i_1, \cdots, i_n)$, and similarly for Bob. For our proofs, we require the definition of elusive range and efficiently verifiable range from Lindell and Pinkas [19].

Definition 1. *Let* $(\mathsf{G},\mathsf{E},\mathsf{D})$ *be a symmetric key encryption scheme with key-generation, encryption, and decryption algorithms. Denote the range of the scheme by* $\mathsf{Range}_n(k) = \{E_k(x)\}_{x \in \{0,1\}^n}$. *We say that*

1. $(\mathsf{G},\mathsf{E},\mathsf{D})$ *has an* ***elusive range*** *if for every probabilistic poly-time machine A, every polynomial $p(\cdot)$, and all sufficiently large n,* $\Pr_{k \leftarrow \mathsf{G}(1^n)}[A(1^n) \in \mathsf{Range}_n(k)] < 1/p(n)$.
2. $(\mathsf{G},\mathsf{E},\mathsf{D})$ *has an* ***efficiently verifiable range*** *if there exists a probabilistic poly-time machine M such that $M(1^n, k, c) = 1$ if and only if $c \in \mathsf{Range}_n(k)$.*

Claim 1. *If the encryption scheme has an elusive range and the oblivious transfer protocol is secure, then Protocol 1 is correct for semi-honest Alice and Bob.*

Proof: First we show that every node in the garbled OBDD sent to Bob can be evaluated correctly. For an encrypted node v, let c_0 and c_1 be the first and second ciphertext term. Suppose k is the key that Bob obtains to decrypt the ciphertext in node v. Because the encryption scheme is elusive and all the keys used in the garbled OBDD are chosen uniformly and independently at random, it follows

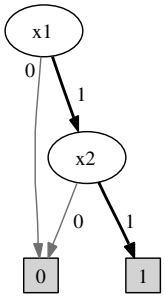

Figure 4(a)

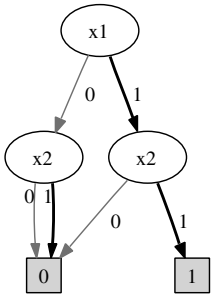

Figure 4(b)

Figure 4: Adding dummy vertices

immediately that, except with negligible probability, only one ciphertext in an encrypted node decrypts correctly; that is, either $c_0 \in \mathsf{Range}(\mathsf{k})$ or $c_1 \in \mathsf{Range}(\mathsf{k})$ but not both. Hence, except with negligible probability, every node in the garbled OBDD can be evaluated correctly.

By induction, we show that the correct key is obtained for every node input and output. The base case is the key $s_{v_{init}}$ and the keys $s_{k+1}^{i_{k+1}}, \ldots, s_n^{i_n}$ (corresponding to Bob's input) obtained from Alice through the $n-k$ oblivious transfers. The inductive step at node v assumes that the correct label (pointing Bob to node v) and node key s_v was obtained in the previous step. Furthermore, the correct level key $s_l^{i_l}$ was obtained by executing an oblivious transfer protocol. Since every node in the garbled OBDD can be evaluated correctly and the garbled OBDD is built by a semi-honest Alice, Bob obtains the correct label and key output at node v, which concludes the inductive step. We can conclude that the entire garbled OBDD can be evaluated to give the correct result.[2] ■

Claim 2. *If the encryption scheme is semantically secure and has an efficiently verifiable elusive range, and the oblivious transfer protocol is secure, then Protocol 1 is secure against semi-honest Alice and Bob.*

Proof of this claim is tedious and is given in Appendix A.

3.2 Protocol 2

The protocol presented in this section allows both Alice and Bob to possess arbitrary input sets instead of assuming that Alice (and also Bob) holds either the first k or the last $n-k$ input variables. In this new protocol, before garbling the OBDD, Alice can reduce the size of the OBDD by eliminating vertices whose labels correspond to Alice's inputs. Let $f(x_1, \cdots, x_n)$ be the function to be computed and X_A denotes the inputs of Alice. Alice first computes the OBDD for the restriction $f\mid_{X_A}$ of f for the variables in its input set X_A. Alice then encrypts the reduced OBDD and sends it to Bob. During the restriction operation, all vertices whose labels correspond to Bob's input should be included. If this is not the case, then there is a risk that different restrictions could produce different numbers of nodes, which would leak information to Bob about Alice's inputs. For example, in

[2]Note that the negligible probability of error during decryption can be removed by Alice first checking that every encrypted node decrypts correctly before sending the garbled OBDD to Bob.

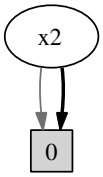

Figure 5(a)

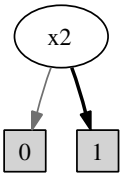

Figure 5(b)

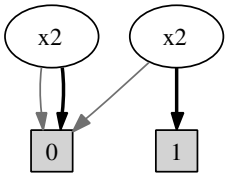

Figure 5(c)

Figure 5: OBDD restriction

Input: Both parties' inputs include the $OBDD(f)$ for the Boolean function $f(x_1, x_2, \cdots, x_n)$ with the ordering $x_1 < x_2 < \cdots < x_n$. Furthermore, Alice holds the inputs for the variables in the set X_A and Bob holds the inputs for the variables in the set $X_B = \{x_1, \cdots, x_n\} - X_A$.

1. Alice performs the following steps:
   (a) Alice computes the OBDD $\mathcal{O}_A$ for the function $f\mid_{X_A}$. This is the restriction operation described in Section 2.
   (b) Alice encrypts the OBDD for the function $f\mid_{X_A}$ and sends it to Bob. This step is exactly the same as for Protocol 1 described in Figure 3. Alice also sends the secret corresponding to the root of the OBDD $\mathcal{O}_A$.
2. The computation for Bob is exactly the same as that for Protocol 1.

Figure 6: Protocol 2.

Figure 5(a), where Alice's value is 0, there are only 2 nodes, but in Figure 5(b), there are 3 nodes. Figure 5(c) shows the result of retaining extra vertices, in which case there are 4 nodes regardless of Alice's inputs. The description of this protocol is given in Figure 6. The proof of correctness of this protocol is almost identical to the one presented in Section 3.1. Example 2 shows an execution of this protocol on a small function.

Example 2. *Assume that Alice and Bob want to compute $f(x_1, x_2) = x_1 \wedge x_2$, where Alice has input x_1 and Bob has input x_2, or in other words $X_A = \{x_1\}$ and $X_B = \{x_2\}$. Assume that $x_1 = 0$ and $x_2 = 1$. $OBDD(f)$ with dummy nodes is shown in figure 4(b). Alice computes the OBDD for the function $f\mid_{x_1 \leftarrow 0}$, which results in a structure shown in Figure 5(c). Let the two nonterminal nodes in Figure 5(c) be v_1 and v_2. First, Alice generates 2 secrets s_{v_1} and s_{v_2}, which are assigned to the nodes v_1 and v_2, respectively. Alice also generates random labels for the four nodes in Figure 5(c), and generates a pair of secrets (s_1^0, s_1^1). The garbled OBDD corresponding to Figure 5(c) is shown below (terminal nodes are shown as 0 and 1 and lab denotes label).*

$(lab(v_1), E_{s_{v_1} \oplus s_1^0}(lab(0) \,\|\, s_0), E_{s_{v_1} \oplus s_1^1}(lab(0) \,\|\, s_0))$
$(lab(v_2), E_{s_{v_2} \oplus s_1^0}(lab(0) \,\|\, s_0), E_{s_{v_2} \oplus s_1^1}(lab(1) \,\|\, s_1))$
$(0, lab(0))$
$(1, lab(1))$

Alice reveals the secret s_{v_1} corresponding node v_1. Alice and Bob engage in a 1-out-of-2 (OT_1^2) protocol and Bob obtains the secret s_1^1 (recall that $x_2 = 1$). Bob can now decrypt the second component of the first entry of the garbled OBDD and obtain $label(0) \,\|\, s_0$, and Bob can infer that the output is 0.

CLAIM 3. *If the encryption scheme has an elusive range and the oblivious transfer protocol is secure, then Protocol 2 is correct for semi-honest Alice and Bob.*

Proof: The proof of this claim is exactly same as the proof of Claim 1. One has to assume that the restriction operation used by Alice is correct. ■

CLAIM 4. *If the encryption scheme is semantically secure and has an efficiently verifiable elusive range, and the oblivious transfer protocol is secure, then Protocol 2 is secure against semi-honest Alice and Bob.*

Proof of this claim is tedious and is given in Appendix A.

4. IMPLEMENTATION AND EXPERIMENTAL RESULTS

In this section we describe various components of our implementation (called SFE-OBDD), which is based on protocol 2 described in Section 3.2. We also present experimental results comparing the performance of our implementation against Fairplay. The following major conclusions can be drawn from our experimental investigation:

- The restriction operation used in protocol 2 can significantly reduce the size of the OBDD, which can lead to reduced bandwidth while executing the protocol.
- Our OBDD protocol outperforms Fairplay circuit in terms of bandwidth in most functions in our benchmark. However, some functions which have inefficient OBDD representations can perform far worse.
- Execution times of our implementation and Fairplay are dominated by the oblivious-transfer protocol. Since the oblivious-transfer component of our protocol and the protocol used by Fairplay is the same, with respect to execution times we did not observe as much improvement as in bandwidth.
- Converting an OBDD into a circuit results in a blowup in size. We implemented a reverse compiler that takes an OBDD and converts it into a circuit description, which can be used in Fairplay. Typically this conversion from OBDD to circuit resulted in a blowup in size by a factor of $5-10$, depending on post-conversion optimizations. However, if the original circuit is particularly inefficient, it is possible for some gain to be achieved due to the canonical representation property of OBDDs.

4.1 Implementation

Our implementation consists of the following components:

1. An implementation of protocol 2 as described in Section 3.2.
2. Fairplay uses the *secure hardware definition language (SHDL)* to describe circuits. Because we wanted to compare the performance of our protocol with the Yao circuit protocol on identical functions, we implemented an OBDD compiler that takes as input a file describing a function in SHDL, and produces the corresponding OBDD. Note that both the SHDL used by Fairplay and the BDD representation originate from the same high level SFDL description, this means that the OBDD and circuit are evaluating the same functions.

For the cryptographic primitives we use exactly the same implementation as Fairplay. We use the 1-out-of-2 (OT_1^2) proposed by Noar and Pinkas [23], and the encryption function is $E_k(m)$ was $\text{SHA}-1(k) \oplus (m \,\|\, 0^n)$.

Our OBDD compiler allows us to directly compare the efficiency of our implementation to Fairplay. The OBDD compiler takes as input a file containing an SHDL description and produces a file containing the description of the corresponding OBDD. This file can then be used as an input to the SFE protocol. Our compiler uses the JavaBDD [16], and BuDDy [5] libraries, which provide functions to construct and manipulate OBDDs. It is well known that the size of an OBDD can be sensitive to the ordering of variables [3]. In some cases, variable ordering can make the difference between a OBDD that is linear versus exponential in the number of variables. Our SHDL to OBDD compiler allows the user to specify a particular variable ordering, which is useful if the user has domain knowledge about the function. If this is not practical, the compiler includes an optimizer that attempts to automatically find a variable ordering that yields an efficient OBDD, making use of heuristic functions built into the BuDDy library. Although in general finding the optimal variable ordering is NP-hard [2], we have found that in practice the optimizer can find good orderings for various functions we considered.

4.2 Experimental Results

We used various functions, some of which are included in the Fairplay distribution, as test cases to perform a comparison of the Fairplay protocol with our OBDD-based SFE protocol. The description of the functions are given in Figure 7. Each function was evaluated at several word sizes to evaluate scalability.

For each function, Figure 8 shows the sizes of the OBDDs and the corresponding circuit used by Fairplay. The size of an OBDD is the number of vertices in it. The size of a Fairplay circuit with n_1 gates of arity 1, n_2 gates of arity 2, and n_3 gates of arity 3 was computed as $2 \times n_1 + 4 \times n_2 + 8 \times n_3$ (this represents the number of entries in the truth table for the circuit). For the OBDDs, we show the sizes of the original OBDDs (in column marked as Original), with the dummy nodes added (in column marked as Full), and after Alice has performed the restriction operation on OBDDs with dummy nodes (in column marked as Res). Recall that dummy nodes are added so that regardless of Alice's inputs Bob has to follow the same number of edges. In protocol 2 Alice computes the OBDD for restriction on its input of the function to be jointly computed for the variable. These operations are described in detail in Section 3. Two observations can be made from Figure 8.

- Restriction can significantly reduce the size of the OBDD. For example, for the function Mil16 restriction reduces the size of the OBDD by more than half.
- Notice that for all functions except MUL8, MUL16, KDS4, KDS8, and KDS16 the size of the OBDD after restriction is smaller than the size of the circuit used in Fairplay. This suggests the choice of when to use our system over Fairplay depends on the function to be computed.

Our experimental results were obtained using a pair of machines connected on a local 100-megabit network. The

And	This is a circuit that computes the bitwise AND of two N bit numbers. Alice has N inputs, Bob has N inputs, and there are N bits of output.
Add	This is a circuit that computes the addition of two N bit numbers. Alice has N inputs, Bob has N inputs, and there are N bits of output. The high bit is discarded.
Eq	This is a simple equality comparator of two N-bit numbers. There is one bit of output.
Mul	This is a circuit that computes the unsigned multiplication of two N bit numbers. Alice has N inputs, Bob has N inputs, and there are N bits of output. The output is modulo 2^N. We were unable to test N=16 because of exponential blowup in the BDD
KDS	This is a circuit that implements a simply-keyed database lookup. Alice supplies N key/value pairs, and Bob supplies a key. The output is the value of that key, or 0 if the key is not found. The keys are $\texttt{log}_2(N)$ bits, and the data are 24 bits. We were unable to test N=16 because of exponential blowup in the BDD.
Mil	This is the millionaire's problem. Alice and Bob each have and N bit integer as inputs, and there is 1 bit of output indicating if Alice's input is larger than Bob's.
Parity	Alice and Bob each have N-bits of input. They want to jointly compute the parity of their combined input bits. There is one bit of output.

Figure 7: Description of the functions used in our experiments. Each function was tested with N=4, N=8, and N=16 except where indicated

	BDD			FairPlay
	Original	Full	Res	
Add4	32	40	22	56
Add8	72	96	54	128
Add16	152	208	118	272
And4	14	18	10	24
And8	26	34	18	48
And16	50	66	34	96
Eq4	18	24	11	102
Eq8	27	41	18	230
Eq16	51	81	34	486
KDS4	416	578	466	356
KDS8	4084	7149	6283	780
KDS16	*	*	*	2244
MUL4	54	75	28	114
MUL8	1685	1800	1087	586
MUL16	*	*	*	2682
Mil4	24	34	22	52
Mil8	46	70	40	116
Mil16	94	150	90	244
parity4	18	18	10	30
parity8	34	34	18	62
parity16	66	66	34	126

Figure 8: Size of the OBDDs and the circuit used in Fairplay for functions shown in Figure 7. Values labeled "*" could not be converted to OBDDs because of exponential blowup.

machines were configured with 3.0Ghz Intel Pentium4 processors, 1 gigabyte of memory, and the Centos Linux 4.0 operating system using a modified Linux 2.6.9 kernel. For each function shown in Figure 7 we executed our OBDD-based and Fairplay code on a Sun Microsystems Java 1.5.0_04 JVM. Alice was run on one machine, and Bob on the other. For each execution, we measured the network bandwidth used (number of bytes transferred between Alice and Bob) and the execution time. The number reported for each trial is the average of three trials. Figure 9 shows the size of the garbled OBDD and garbled circuit in bytes and the network bandwidth for our implementation and Fairplay. Recall that the garbled OBDD is the structure that Alice sends to Bob to evaluate. With respect to network bandwidth our implementation outperformed Fairplay for seven out of the nine functions. We have implemented a reverse compiler that takes as input an OBDD, and outputs an SHDL description of a boolean circuit to evaluate the OBDD. This is performed via a straightforward transformation that takes each node in the OBDD and produces corresponding 3-input MUX gate in the boolean circuit. Then, an optimization pass is run using the same techniques described in [22]. The column labeled as "Converted Fairplay" in Figure 9 shows the size of the encrypted circuits produced by running the FairPlay protocol on the converted BDDs. Note that in a few cases, the converted circuit is actually more efficient than the corresponding Fairplay circuit. This occurs because FairPlay is not guaranteed to produce an optimal circuit from the function description. However, it is clear that our protocol that directly uses OBDD is much more efficient than the protocol produced by the reverse compiler.

Figures 10 and 11 show the execution times for SFE-OBDD and Fairplay. The elapsed execution times (EET) are shown in the last column. Columns 2-5 show the breakdown by sub-task, which are IPCG (initializations, parsing, and garbling), CC (circuit communication, Alice sending the garbled structure to Bob), OT (Oblivious Transfer, Bob obtaining secrets corresponding to its input), and EV (circuit evaluation, Bob evaluating the garbled structure). These sub-tasks were also used by the Fairplay paper [22]. In general, because the time for OT dominates the execution time, we only observe moderate improvement in SFE-BDD over Fairplay for execution times.

Function	Size in bytes.			Bandwidth in bytes	
	SFE-OBDD	Fairplay	Converted Fairplay	SFE-OBDD	Fairplay
Add4	970	1915	5382	3604	4684
Add8	1979	4214	11408	6590	9000
Add16	3992	8821	23442	12557	17645
And4	739	1080	1866	3373	3849
And8	1206	2153	3140	5813	6938
And16	2134	4299	5546	10696	13117
Eq4	582	2977	2690	3214	5716
Eq8	828	6527	3918	5434	11240
Eq16	1324	13626	7012	9892	22295
KDS4	14966	12248	65946	51219	13984
KDS8	185282	25608	682333	261077	27838
MUL4	1108	3286	8354	3739	6052
MUL8	32206	15706	288317	36814	20493
Mil4	892	1662	4278	3524	4399
Mil8	1400	3542	8248	6012	8256
Mil16	2790	7306	16859	11356	15972
parity4	577	1092	3237	3209	3830
parity8	828	2181	6172	5438	6897
parity16	1324	4359	11983	9889	13033

Figure 9: Size in bytes of the garbled OBDD, garbled circuit, and garbled circuit using the reverse compiler. Network bandwidth in bytes.

Fn	IPCG	CC	OT	Eval	EET
Add4	13.75%	5.62%	79.69%	0.94%	0.32
Add8	13.08%	3.80%	82.07%	1.05%	0.47
Add16	11.39%	2.36%	84.42%	1.83%	0.76
And4	12.42%	5.73%	81.21%	0.64%	0.31
And8	9.38%	4.02%	85.94%	0.67%	0.45
And16	7.44%	2.75%	89.39%	0.41%	0.73
Eq4	12.66%	5.38%	81.33%	0.63%	0.32
Eq8	9.33%	3.90%	86.12%	0.65%	0.46
Eq16	8.97%	2.48%	88.14%	0.41%	0.72
KDS4	4.79%	0.95%	93.86%	0.40%	2.52
KDS8	10.91%	1.69%	87.13%	0.27%	5.50
MUL4	14.77%	5.23%	79.38%	0.62%	0.33
MUL8	31.80%	4.11%	63.45%	0.63%	0.63
Mil4	13.44%	5.62%	80.00%	0.94%	0.32
Mil8	11.37%	3.86%	84.12%	0.64%	0.47
Mil16	10.55%	3.03%	85.88%	0.53%	0.76
parity4	10.67%	4.78%	83.71%	0.84%	0.36
parity8	9.37%	3.92%	86.06%	0.65%	0.46
parity16	8.55%	2.62%	88.41%	0.41%	0.72

Figure 10: Elapsed execution time (EET) in seconds and their breakdowns into sub-tasks for SFE-OBDD.

Fn	IPCG	CC	OT	Eval	EET
Add4	17.65%	19.00%	63.12%	0.23%	0.44
Add8	16.13%	15.96%	67.38%	0.53%	0.56
Add16	10.74%	9.67%	79.12%	0.48%	0.84
And4	11.92%	20.44%	67.40%	0.24%	0.41
And8	11.78%	16.07%	71.96%	0.19%	0.54
And16	9.78%	6.35%	83.48%	0.38%	0.79
Eq4	19.51%	11.85%	68.15%	0.49%	0.41
Eq8	15.44%	16.52%	67.68%	0.36%	0.56
Eq16	13.23%	9.84%	76.70%	0.23%	0.85
KDS4	33.33%	12.75%	53.33%	0.58%	0.34
KDS8	35.98%	11.92%	51.43%	0.66%	0.45
MUL4	21.89%	2.16%	75.41%	0.54%	0.37
MUL8	21.75%	7.99%	69.89%	0.37%	0.54
Mil4	38.27%	8.26%	53.28%	0.19%	0.53
Mil8	16.98%	9.25%	73.40%	0.38%	0.53
Mil16	18.78%	9.01%	71.99%	0.22%	0.92
parity4	20.78%	11.25%	67.73%	0.24%	0.41
parity8	49.55%	0.39%	49.94%	0.13%	0.77
parity16	14.63%	1.15%	83.97%	0.25%	0.79

Figure 11: Elapsed execution time (EET) in seconds and their breakdowns into sub-tasks for Fairplay.

5. FUTURE WORK

There are other optimizations to OBDDs that we have not explored in this paper. For example, adding negated edges to OBDDs can result in smaller structures for some Boolean functions.[3] Incorporating these optimizations in our protocol while preserving privacy is a direction for future work. There are several other OBDD-like representations developed by the computer-aided design and computer aided verification research communities, such as Binary Moment Diagrams (BMDs) [4] and Hybrid Decision Diagrams (HDDs) [8]. For a certain class of functions, these representations are more succinct than OBDDs. For example, BMDs can efficiently represent integer multiplication, which cannot be represented efficiently at the bit-level with OBDDs. Extending our protocol for these representations is an important direction of future research. Our vision is to provide an option for all these representations in our system so that a user can choose the representation that is suitable for the problem.

OBDDs have been used for a variety of applications, such efficient filtering in publish-subscribe systems [6], program analysis [26], and planning [17]. In the future we will investigate whether our protocol can be extended to design privacy-preserving algorithms for these applications.

6. REFERENCES

[1] Gagan Aggarwal, Nina Mishra, and Benny Pinkas. Secure computation of the k-th ranked element. In Christian Cachin and Jan Camenisch, editors, *Proceedings of Eurocrypt 2004*, volume 3027 of *LNCS*, pages 40–55. Springer-Verlag, May 2004.

[2] Beate Bollig and Ingo Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, 45(9), September 1996.

[3] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[4] Randal E. Bryant and Yirng-An Chen. Verification of arithmetic circuits with binary moment diagrams. In *Proceedings of the 32nd Conference on Design Automation (DAC)*, 1995.

[5] Buddy. `http://sourceforge.net/projects/buddy`.

[6] A. Campailla, S. Chaki, E. M. Clarke, S. Jha, and H. Veith. Efficient filtering in publish-subscribe systems using binary decision diagrams. In *Proceedings of the 23rd International Conference on Software Engineering (ICSE)*, 2001.

[7] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. The MIT Press, 2000.

[8] E.M. Clarke, M. Khaira, and X. Zhao. Hybrid decision diagrams: Overcoming the limitations of MTBDDs and BMDs. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, 1995.

[9] Lorrie Cranor, Marc Langheinrich, Massimo Marchiori, Martin Presler-Marshall, and Joseph Reagle. *The Platform for Privacy Preferences 1.0 (P3P1.0) Specification*. W3C Recommendation, 16 April 2002.

[3]Following a negated edge in an OBDD flips the value of the result.

[10] Lorrie Faith Cranor. Internet privacy. *Communications of the ACM*, 42(2):28–38, 1999.

[11] J. Feigenbaum, B. Pinkas, R. Ryger, and F. Saint-Jean. Secure computation of surveys. In *2004 EU Workshop on Secure Multiparty Protocols (SMP)*, 2004.

[12] Michael Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In Christian Cachin and Jan Camenisch, editors, *Proceedings of Eurocrypt 2004*, volume 3027 of *LNCS*, pages 1–19. Springer-Verlag, May 2004.

[13] Ian Goldberg, David Wagner, and Eric Brewer. Privacy-enhancing technologies for the internet. In *Proc. of 42nd IEEE Spring COMPCON*. IEEE Computer Society Press, February 1997.

[14] O. Goldreich. *The Foundations of Cryptography — Volume 2*. Cambridge University Press, 2004.

[15] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game – a completeness theorem for protocols with honest majority. In *19th STOC*, pages 218–229, 1987.

[16] Javabdd - java binary decision diagram library. `http://javabdd.sourceforge.net/`.

[17] R. M. Jensen and M. M. Veloso. Obdd-based universal planning for multiple synchronized agents in non-deterministic domains. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS)*, 2000.

[18] Y. Lindell and B. Pinkas. Privacy preserving data mining. *Journal of Cryptology*, 15(3), 2002.

[19] Yehuda Lindell and Benny Pinkas. A proof of Yao's protocol for secure two-party computation. Cryptology ePrint Archive, Report 2004/175, 2004. `http://eprint.iacr.org/2004/175`.

[20] B. Pinkas M. Naor and R. Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM conf. on Electronic Commerce*, 1999.

[21] Philip D. MacKenzie, Alina Oprea, and Michael K. Reiter. Automatic generation of two-party computations. In *Proceedings of ACM Conference on Computer and Communications Security*, 2003.

[22] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — a secure two-party computation system. In *Proceedings of the 13th Usenix Security Symposium*, San Diego, CA, USA, August 2004.

[23] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms (SODA)*, pages 448–457, 2001.

[24] D. M. Rind, I. S. Kohane, P. Szolovits, C. Safran, H. C. Chueh, and G. O. Barnett. Maintaining the confidentiality of medical records shared over the internet and the world wide web. *Annals of Internal Medicine*, 127(2), July 1997.

[25] Joseph Turow. Americans and online privacy: The system is broken. Technical report, Annenberg Public Policy Center, June 2003.

[26] J. Whaley and M. S. Lam. Cloning-based context-sensitive pointer alias analysis using binary decision diagrams. In *Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation (PLDI)*, 2004.

[27] A.C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, 1986.

APPENDIX

A. PROOF OF CORRECTNESS

Proof: [*Proof of Claim 2*] Intuitively, Bob's security follows directly from the security of the 1-out-of-2 oblivious transfer protocol he uses to obtain the secrets corresponding to his input. Alice's security follows from both the security of the oblivious transfer protocol (allowing Bob to only obtain only one key per node) and the semantic security of the encryption scheme (which allows Bob to only decrypt one entry in each node). We now flesh out the details by providing a simulation proof from Alice and Bob's view of the protocol. Let x and y be the inputs of Alice and Bob respectively and Π be a protocol for secure-function evaluation of $f(x,y)$. Let $\mathsf{VIEW}_A^\Pi(x,y)$ and $\mathsf{VIEW}_B^\Pi(x,y)$ be the view of Alice and Bob for the run of the protocol Π on input x and y (view of a party consists of its input, output, and all messages it receives during the execution of the protocol). In a simulation proof one needs to show two probabilistic polynomial-time algorithms S_A and S_B such that $S_A(x, f(x,y))$ and $S_B(y, f(x,y))$ are computationally indistinguishable from $\mathsf{VIEW}_A^\Pi(x,y)$ and $\mathsf{VIEW}_B^\Pi(x,y)$, respectively. For a precise definition of a simulation proof the reader should refer to [14, Chapter 7].

We first consider the case where Alice is corrupt. Alice's view in an execution of Protocol 1 consists of her view of the oblivious transfer protocol executions and the output of the function from Bob at the end. We now build a simulator that simulates Alice's view given access only to her input and output. Because the oblivious transfer protocol is secure, there exists a simulator that can simulate the transcript of Alice's view of the oblivious transfer protocol without knowing Bob's input. On input $(i_1, \ldots, i_k, f(i_1, \ldots, i_n))$, the simulator first simulates Alice's view of all $n-k$ executions of the oblivious transfer protocol by repeatedly running the oblivious transfer protocol simulator. Using a standard hybrid argument on the transcripts of all $n-k$ executions oblivious transfer protocols, we see that if the oblivious transfer protocol is secure, then the distributions of the simulated and real combined transcripts of all $n-k$ oblivious transfer executions are indistinguishable with non-negligible probability.

Finally, the simulator writes $f(i_1, \ldots, i_n)$ on the transcript of Alice's view. We now show that the distribution of the output f is indistinguishable (except with negligible probability) from the real output, which amounts to showing that Bob outputs $f(i_1, \ldots, i_n))$ correctly on a real interaction. By the security of the oblivious transfer protocol, Bob is provided with the correct keys corresponding to its input during each execution of the oblivious transfer protocol. Applying claim 1, it follows immediately that, except with negligible probability, Alice obtains the correct output from Bob, except with negligible probability. Therefore, the distribution of the simulated transcript is indistinguishable, except with negligible probability, from a real transcript, concluding the case when Alice is corrupt.

We now consider the case when Bob is corrupt. Given $(i_{k+1}, \ldots, i_n, f(i_1, \ldots, i_n))$, the simulator $\mathcal{S}$ must simulate both a garbled OBDD that Bob can use to correctly compute $f(i_1, \ldots, i_n)$, and Bob's view of the $n-k$ executions of the oblivious transfer protocol. We first show how $\mathcal{S}$ simulates the $n-k$ oblivious transfer protocol executions. As in the previous case, because the oblivious transfer protocol is secure, there exists a simulator that can simulate the transcript of Bob's view of the oblivious transfer protocol without knowing Alice's input. Therefore, $\mathcal{S}$ simulates Bob's view of all $n-k$ executions of the oblivious transfer protocol by running the oblivious transfer protocol simulator $n-k$ times. Using a standard hybrid argument on the transcripts of the oblivious transfer protocols, we see that if the oblivious transfer protocol is secure, then the distributions of the simulated and real transcripts of all $n-k$ oblivious transfer executions are indistinguishable except with negligible probability.

We now show how $\mathcal{S}$ builds a garbled OBDD that Bob can use to successfully compute $f(i_1, \ldots, i_n)$. Since $\mathcal{S}$ does not know $i_1, \ldots, i_k$, it cannot generate the garbled OBDD according to the protocol instructions. Instead, $\mathcal{S}$ generates a garbled OBDD that always evaluates to $f(i_1, \ldots, i_n)$ regardless of the keys used. Such a garbled OBDD is built by first generating a chain of $n-k$ garbled nodes $n_{k+1}, \ldots, n_n$ such that Bob's computation starts at n_{k+1} and proceeds along the chain through n_{k+2} and so on, before ending at node n_n; note that there is one such node for every level from $k+1$ to n. To ensure the computation always proceeds along this chain, *both* ciphertexts in garbled nodes $n_{k+1}, \ldots, n_{n-1}$ are encryptions (under different keys) of the same label-key message such that the label points to the next node along the chain and the node key combined with the level key allows successful decryption of that node; for example, simulated node n_j for $k+1 \leq j \leq n-1$ has the form

$$\Big(label(n_j)\ ,\ E_{s_{n_j} \oplus s_l^0}(label(n_{j+1}) \parallel s_{n_{j+1}}) \\ E_{s_{n_j} \oplus s_l^1}(label(n_{j+1}) \parallel s_{n_{j+1}})\Big)\ .$$

Node n_n is the terminal node and it is set to $f(i_1, \ldots, i_n)$. Once $n_{k+1}, \ldots, n_n$ is generated, the simulator generates a number of "fake" nodes so that the simulated garbled OBDD contains the correct number of nodes; this number can be determined from $OBDD(f)$. Fake nodes are nodes whose ciphertext pair contain encryptions under different keys of the same label-key message; in a fake node, the label, the keys used to encrypt the ciphertext pair, and the label-key message encrypted in the ciphertext pair are chosen randomly.

All that remains is to show that the distribution of the simulated garbled OBDD is indistinguishable from that of a real garbled OBDD. We do this by using a standard hybrid argument over the nodes in the garbled OBDD. Specifically, we run hybrid experiments with garbled OBDDs where real nodes are replaced by simulated nodes. We define the hybrid distributions such that $H_0(i_1, \ldots, i_n)$ contains the real garbled OBDD and $H_B(i_1, \ldots, i_n)$ contains the simulated garbled OBDD where B is the number of non-dummy nodes in the real garbled OBDD. We do not need to consider dummy nodes in our hybrid experiment OBDDs because dummy nodes have the same distribution as the simulated fake nodes and do not affect our argument.

We now define the hybrid garbled OBDD in experiment $H_i(i_1, \ldots, i_n)$; the difficulty here is that the hybrid OBDD

contains both real and simulated nodes but must still allow Bob to correctly compute $f(i_1, \ldots, i_n)$. First, we traverse the real garbled OBDD and label a node as active if it is used by Bob in the process of evaluating the OBDD and inactive otherwise. Note that there will be only $n - k$ active nodes. Next, we order the nodes in the garbled OBDD by their level with level $j + 1$ nodes placed ahead of level $j + 2$ and so on; within the same level, nodes are ordered arbitrarily. The hybrid OBDD is defined as follows: first take the real garbled OBDD and replace the first i non-dummy nodes as follows: inactive nodes are replaced with simulated fake nodes. An active node at level j is altered by replacing its current ciphertext pair with two encryptions of the label-key message corresponding to the next active node at level $j + 1$. These replacement ciphertexts are created with the keys used to create the original ciphertext pair. Note that the distribution of this altered active node is identical to that of the simulated node n_j in the node chain described above. It is easy to see that a garbled OBDD built with this definition has the same distribution as 1) a real garbled OBDD when $i = 0$ (i.e. for H_0), and 2) a simulated garbled OBDD when $i = B$ (i.e. for H_B).

We are now ready to show that the distribution of the simulated garbled OBDD is indistinguishable from that of a real garbled OBDD; that is, we will show that $\{H_0(i_1, \ldots, i_n)\} = \{H_B(i_1, \ldots, i_n)\}$. Suppose to the contrary that the distributions are distinguishable; that is, there exists a poly-time distinguisher $\mathcal{D}$ that

$$|\Pr[\mathcal{D}(H_0(i_1, \ldots, i_n)) = 1] - \Pr[\mathcal{D}(H_B(i_1, \ldots, i_n)) = 1] > 1/p|$$

for some polynomial p. Then there exists a j such that

$$|\Pr[\mathcal{D}(H_{j-1}(i_1, \ldots, i_n)) = 1] - \Pr[\mathcal{D}(H_j(i_1, \ldots, i_n)) = 1] > 1/pB|$$

Using $\mathcal{D}$, we now build an adversary that breaks the semantic security of the encryption scheme used to encrypt the garbled nodes. Recall in a semantic security game, the adversary sends two messages m_0, m_1 to the challenger and receives the encryption of m_b for $b = \{0, 1\}$; the adversary's goal is to determine b. Let n^j be the jth node and we denote its two ciphertext terms as c_0^j and c_1^j. Note that node n^j in the hybrid OBDD in distribution H_{j-1}^* is a real garbled node, whereas the same node for distribution H_j^* is a simulated garbled node; specifically, c_0^j and c_1^j in distribution H_{j-1}^* are encryptions of different label-key messages, whereas they are encryptions of the same label-key message in distribution H_j^*. We exploit this fact to build the adversary $\mathcal{A}$ that breaks semantic security of the encryption scheme.

First, $\mathcal{A}$ creates the hybrid garbled OBDD corresponding to the distribution $H_{j-1}^*(i_1, \ldots, i_n)$. One of the two ciphertexts c_0^j and c_1^j in node n^j is an encryption of the label and key for the active node n^{j+1}, whereas the other ciphertext is an encryption of the label and key for an inactive node. Let ℓ_0 be the label-key message encrypted in c_0^j and ℓ_1 be that encrypted in c_1^j. Next, $\mathcal{A}$ sends ℓ_0 and ℓ_1 to the semantic security challenger and receives c^*, which is an encryption of either ℓ_0 or ℓ_1. Without loss of generality, let ℓ_0 be the label-key message (contained in ciphertext c_0^j) that leads to the next active node. $\mathcal{A}$ replaces c_1^j with c^* in node n^j in the garbled OBDD that it built in the first step, and then feeds the altered OBDD together with the other required inputs to the hybrid distinguisher $\mathcal{D}$. Note that c^* cannot be decrypted with the node and level keys for node n^j. This fact, however, does not prevent the garbled OBDD from being evaluated correctly because c^* replaces c_1^j, which contains the label and key to an inactive node, and would not be successfully decrypted while evaluating the garbled OBDD on the inputs $(i_1, \ldots, i_n)$.

$\mathcal{D}$ eventually outputs a result stating that the input is of distribution H_{j-1}^* or H_j^*. If $\mathcal{D}$ outputs that the input is of distribution H_{j-1}^*, then $\mathcal{A}$ outputs that c^* is an encryption of ℓ_1; otherwise $\mathcal{A}$ outputs that c^* is an encryption of ℓ_0. Notice that if c^* is an encryption of ℓ_0, then both ciphertexts in node n^j are encryptions of the same label-message, and the input to $\mathcal{D}$ has distribution H_j^*. Similarly, if c^* is an encryption of ℓ_1, then the input to $\mathcal{D}$ has distribution H_{j-1}^*. Since $\mathcal{D}$ distinguishes between H_{j-1}^* and H_j^* with non-negligible probability, we see that $\mathcal{A}$ wins the semantic security game with non-negligible advantage. Since we assume that the encryption scheme is semantically secure, this implication is a contradiction, and there is no such distinguisher $\mathcal{D}$ that distinguishes between $H_0(i_1, \ldots, i_n)$ and $H_B(i_1, \ldots, i_n)$; that is, the distribution of the simulated garbled OBDD is indistinguishable from that of a real garbled OBDD. Therefore, Bob's simulated view is indistinguishable, except with negligible probability, to the real view, concluding the proof. ■

Proof: [*Proof Sketch for Claim 4*] The full proof is very similar for that in Claim 2, and we provide only a sketch. The case when Alice is corrupt is identical to that in Claim 2. We briefly discuss why the proof is also almost identical in the case when Bob is corrupt. The main observation is that the simulator $\mathcal{S}$ builds the simulated garbled OBDD in the same way as in Claim 2 because there is no difference in how Protocol 2 requires Bob to traverse the garbled nodes given the same level keys. Therefore, the hybrid distributions are defined the same way and the rest of the proof follows. ■