An Efficient and Secure Boolean Function Evaluation Protocol

Sushmita Sarkar[1], Vikas Srivastava[2*], Tapaswini Mohanty[3], Nibedita Kundu, Sumit Kumar Debnath[1]

[1]Department of Mathematics, National Institute of Technology Jamshedpur, Jamshedpur, 831014, Jharkhand, India.
[2*]Department of Mathematics, Indian Institute of Technology Madras, Chennai, 600036, India.
[3]Department of Computer Science and Enginnering, Indian Institute of Technology Roorkee, Roorkee, 247667, India.

*Corresponding author(s). E-mail(s): vikas.math123@gmail.com;
Contributing authors: sarkarsushmita408@gmail.com; mtapaswini37@gmail.com; nknkundu@gmail.com; sd.iitkgp@gmail.com;

Abstract

Boolean functions play an important role in designing and analyzing many cryptographic systems, such as block ciphers, stream ciphers, and hash functions, due to their unique cryptographic properties such as nonlinearity, correlation immunity, and algebraic properties. The secure evaluation of Boolean functions or Secure Boolean Evaluation (SBE) is an important area of research. SBE allows parties to jointly compute Boolean functions without exposing their private inputs. SBE finds applications in privacy-preserving protocols and secure multi-party computations. In this manuscript, we present an efficient and generic two-party protocol (namely BooleanEval) for the secure evaluation of Boolean functions by utilizing a 1-out-of-2 Oblivious Transfer (OT) as a building block. BooleanEval only employs XOR operations as the core computational step, thus making it lightweight and fast. Unlike other lightweight state-of-the-art designs of SBE (such as [1] and [2]), BooleanEval avoids the use of additional cryptographic primitives, such as hash functions and commitment schemes to reduce the computational overhead.

Keywords: Boolean Functions, Secure Boolean Function Evaluation, Oblivious Transfer, XOR, Semi-Honest Security

1 Introduction

A Boolean function is a mathematical function that takes binary inputs and produces a binary output. They are a fundamental component [3] in many cryptographic systems. In particular, Boolean functions have applications ranging from block ciphers [4] and stream ciphers [4] to hash functions [5] and digital signatures. The cryptographic properties of Boolean functions, such as nonlinearity, resiliency, and balance, make them important for building cryptographic algorithms that resist various attacks. The secure evaluation of Boolean functions (also known as **S**ecure **B**oolean **E**valuation (SBE)) is an active and critical area of research. The research in this domain is driven by its wide-ranging applications in privacy-preserving protocols and secure multi-party computations. A two-party SBE allows two parties, Alice and Bob, to jointly evaluate a Boolean function without revealing their respective inputs. The ability to securely compute Boolean functions implies the ability to evaluate any computable function securely (also known as **S**ecure **F**unction **E**valuation (SFE) in literature) [6]. No matter how complex, any function can be represented as a composition of Boolean functions. Every function can be broken down into a series of logical operations, such as AND, OR, and NOT, which form the building blocks of Boolean circuits. Summing up, to securely evaluate a computable function f, we represent it through its corresponding boolean circuit C. In the following, SBE is employed to evaluate C securely. SBE has significant applications in multi-party computation, where privacy is one of the major concerns. For example, it has applications in domains such as privacy-preserving biometric authentication [7, 8], secure deep learning framework [9], privacy-preserving machine learning [10], and medical emergencies [11]. Besides, database mining and data storage outstanding [12, 13] are some of the important applications of SBE. SBE is also a universal building block, and many interesting cryptographic protocols can be formulated as instances thereof, e.g., zero-knowledge proofs.

The first SBE protocol [14] was proposed by Yao as a solution to the famous millionaires' problem [15]. The problem states the following. Two millionaires, X and Y, who possess wealth amounts x and y respectively, aim to determine who is wealthier without disclosing their actual wealth. Yao proposed an SFE protocol as a solution, using a function f such that $f(x, y) = 1$ if Y is wealthier than X, and 0 otherwise. The goal is to compute $f(x, y)$ while preserving the privacy of both parties' wealth. In the current state-of-the-art, there have been several constructions of SBE based on various techniques such as garbled circuit [1, 2, 14, 16], mix and match [17], and server-aided [18]. In this work, we focus on designing and analyzing a protocol for the secure evaluation of Boolean functions using simple cryptographic techniques such as Oblivious Transfer (OT) and XOR as building blocks. We briefly review the existing state-of-the-art protocols [1, 2] that utilize the same idea. Malkhi et al. [1] introduced Fairplay, a full-fledged system that enables generic secure function evaluation (SFE). It is a two-party SFE protocol where one party, Alice, garbles circuits using the hash function and XOR operation. Another party, Bob, chooses one of those circuits to evaluate and verify the remaining using the commitment scheme. In the end, they are involved in 1-out-of-2 OT to get the output. The complexity of this protocol depends on the size of the circuit, the number of circuits garbled, and the commitment scheme and OT used in the protocol. Kolesnikov et al. [2] designed a two-party SFE protocol by garbling the circuit corresponding to the function that has to be evaluated. One party (say Alice) uses a random oracle and simple XOR operation to execute the process. Then, Alice and Bob are involved in 1-out-of-2 OT to send the necessary information to Bob to compute the output. Bob evaluates the garbled circuit and obtains the output.

Our contributions. The major contributions of this paper are mentioned below.

- In this manuscript, we focus on the design and analysis of a two-party SBE protocol (namely BooleanEval). We present new techniques for secure evaluation of Boolean functions using only standard cryptographic primitives as building blocks. BooleanEval allows two parties, Alice and Bob, to securely compute a Boolean function over their private inputs.
- In particular, we propose a generic design that utilizes 1-out-of-2 OT as a building block. Any secure 1-out-of-2 OT can be used to instantiate the BooleanEval. Apart from OT, the protocol utilizes only XOR operations as the core computational step. XOR operations are efficient and fast. Thus, it leads to minimal computational overhead and rapid execution.
- BooleanEval is efficient when compared to existing state-of-the-art lightweight designs such as [1] and [2]. Unlike [1, 2], BooleanEval does not employ additional cryptographic primitives such as cryptographically secure hash functions and commitment protocols.

Organization of the paper. The preliminaries are contained in Section 2. Our proposed secure Boolean function evaluation design (BooleanEval) is described in Section 3. Security analysis and efficiency analysis are provided in Section 4 and Section 5, respectively. Finally, we have concluded this paper in Section 6.

2 Preliminaries

2.1 1-out-of-2 OT

1-out-of-2 OT is a two-party privacy preserving cryptographic protocol. It involves two parties say Alice and Bob where Alice has two messages (m_0, m_1). Bob is allowed to obtain message m_b according to his choice bit $b \in \{0, 1\}$. A schematic diagram of 1-out-of-2 OT is shown in Figure 1. A 1-out-of-2 OT satisfies the following security properties.

- Alice's privacy: Bob should not be able to obtain messages other than the chosen message m_b.
- Bob's privacy: Alice should be unable to gain any information about the choice bit b of Bob.

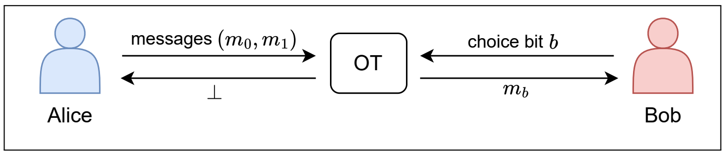


Fig. 1 1-out-of-2 Oblivious Transfer

2.2 Boolean function

A Boolean function f is defined as a map $f : \{0,1\}^{v_1} \longrightarrow \{0,1\}$ for some $v_1 \in \mathbb{N}$. However, the function f can be generalized as a map from v_1 tuple binary string to an u tuple binary string defined as $f : \{0,1\}^{v_1} \longrightarrow \{0,1\}^u$. Some simple example of Boolean functions are given in Example 1, 2, and 3.

Example 1. $f_{\mathsf{AND}} : \{0,1\} \times \{0,1\} \longrightarrow \{0,1\}$ *by*

$$f_{\mathsf{AND}}(X,Y) = X \cdot Y = \begin{cases} 0 & \textit{if one of } X \textit{ or } Y \textit{ is 0} \\ 1 & \textit{if both } X \textit{ and } Y \textit{ are 1} \end{cases}$$

Example 2. $f_{\mathsf{NOT}} : \{0,1\} \longrightarrow \{0,1\}$ *by*

$$f_{\mathsf{NOT}}(X) = \neg(X) = \begin{cases} 0 & \textit{if } X = 1 \\ 1 & \textit{if } X = 0 \end{cases}$$

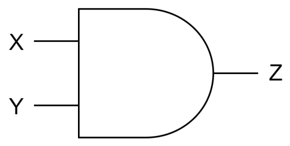
Example 3. $f_{\mathsf{OR}} : \{0,1\} \times \{0,1\} \longrightarrow \{0,1\}$ *by*

$$f_{\mathsf{OR}}(X,Y) = X + Y = \begin{cases} 0 & \textit{if both of } X \textit{ and } Y \textit{ are 0} \\ 1 & \textit{if one of } X \textit{ or } Y \textit{ is 1} \end{cases}$$

2.3 Universal Logic Gates

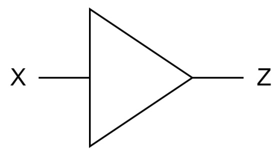
Logic gates are fundamental tool for constructing a digital circuit. Basic logic gates are AND, NOT, OR gates. In general, AND, NOT, OR operations are represented as $\cdot, \neg, +$ respectively. The constructions and truth tables of these gates are given in Example 4, 5, and 6.

Example 4. AND *gate construction and it's truth table.*



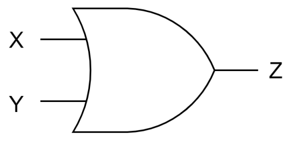
Input		*Output*
X	***Y***	***Z***
0	*0*	*0*
0	*1*	*0*
1	*0*	*0*
1	*1*	*1*

Example 5. NOT *gate construction and it's truth table.*



Input	*Output*
X	***Z***
0	*1*
1	*0*

Example 6. OR *gate construction and it's truth table.*



Input		*Output*
X	***Y***	***Z***
0	*0*	*0*
0	*1*	*1*
1	*0*	*1*
1	*1*	*1*

Note that the truth tables of the logic gates AND, OR, NOT represents the corresponding Boolean function's output (refer Example 1, 2, 3). Moreover, any other logic gate can be constructed using only AND and NOT gates. Therefore, AND, NOT gates together are called universal logic gates. Consequently, any Boolean circuit C can be constructed using AND, NOT logic gates. Any Boolean function f can also be represented as a Boolean circuit C such that the output of the function f and the circuit C are same.

De Morgan's Law. For any two propositions X, Y the De Morgan's law says.

$$\neg(X + Y) = (\neg(X)) \cdot (\neg(Y)) \tag{1}$$
$$\neg(X \cdot Y) = (\neg(X)) + (\neg(Y)) \tag{2}$$

Input and Non-input Gates. Let C be the circuit corresponding to the Boolean function f. We divide the gates in C into two separate classes: input gates and non-input gates. We provide a illustrative example in Figure 2.

- *Input Gate.* The gates G_i whose input wires are not the output of any other gates of that circuit are defined as input gates. For example, gates G_1, G_2, G_3 are input gates in Figure 2.
- *Non-input Gate.* The gates G_i whose one or both of the input wires are output wires of any other gates in C are defined as non-input gates. For instance, gates G_4, G_5, G_6 are non-input gates in Figure 2.

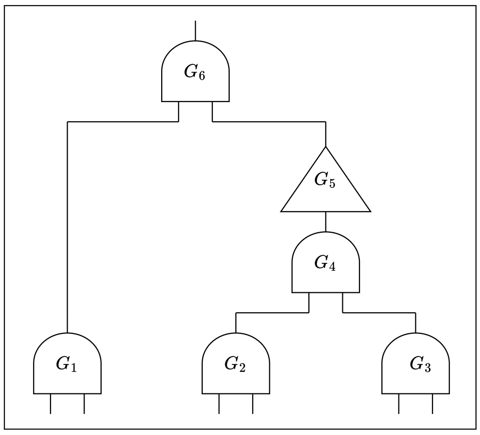


Fig. 2 Example of input and non-input gates of a circuit C

3 Proposed Secure Boolean Function Evaluation Protocol $\mathsf{BooleanEval}$

A high level overview: In this work, we concentrate ourselves in the design of a two-party secure Boolean function evaluation protocol namely $\mathsf{BooleanEval}$. Our scheme involves two parties: sender Alice and receiver Bob having their private inputs $x = (x_1, x_2, \ldots, x_{v_1}) \in \{0,1\}^{v_1}$ and $y = (y_1, y_2, \ldots, y_{v_2}) \in \{0,1\}^{v_2}$ respectively where $v_1, v_2 \in \mathbb{N}$. They wish to securely evaluate a Boolean function $f : \{0,1\}^{v_1} \times \{0,1\}^{v_2} \to \{0,1\}^u$ for some $u \in \mathbb{N}$. Note that any Boolean function f can be represented as an acyclic Boolean circuit C [2] using only AND and NOT gates such that, for all inputs $x \in \{0,1\}^{v_1}, y \in \{0,1\}^{v_2}$, $C(x,y) = f(x,y)$. We are utilizing the fact that AND and NOT gates together is known as universal logic gates. The input AND gates and input NOT gates with respect to the input y_l of Bob are obliviously evaluated by executing an OT protocol between Alice and Bob. On the other hand, for each input NOT gate with respect to input x_k of Alice, an arbitrary bit string $a_{\bar{x}_k}$ is sent to Bob by Alice where $a_0, a_1 \in_R \{0,1\}^\lambda$. In case of OT for input AND gate, Alice's input pair corresponding to its input bit x_k is of the form $(a_0, \bar{x}_k a_0 \oplus x_k a_1)$ and Bob's input choice is its input bit y_l where $a_0, a_1 \in \{0,1\}^\lambda$. While in case of OT for input NOT gate, Alice's input pair is (a_0, a_1) and Bob's input pair is $\bar{y}_l$ for its input bit y_l, where $a_0, a_1 \in_R \{0,1\}^\lambda$. For the case of each non-input AND gate, Alice makes a set containing all four possible outcomes and for each non-input NOT gate, it makes a set containing all two possible outcomes. The outcomes in each case are represented by arbitrary bit strings with the help of associated input wire bit strings. Moreover, these outcomes are written in randomized order in the set. As a result, after execution Bob will get only arbitrary bit strings as output. At the final stage, Alice sends associated bit strings (a, b) for the final output gate as $(a, 0)$ and $(b, 1)$ so that Bob can determine the output as 0 if he gets a and 1 if he gets b.

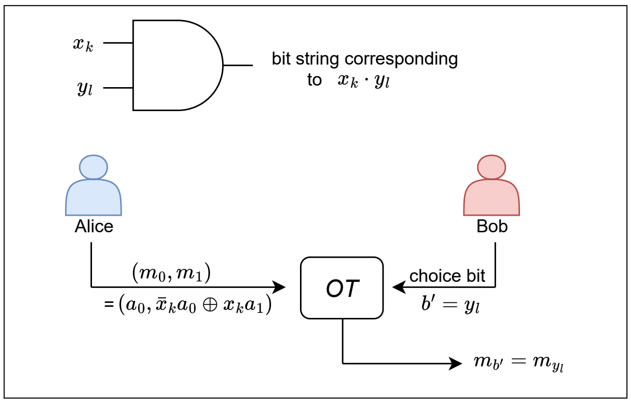

x_k
y_l
bit string corresponding to $x_k \cdot y_l$
Alice
Bob
(m_0, m_1)
$= (a_0, \bar{x}_k a_0 \oplus x_k a_1)$
OT
choice bit
$b' = y_l$
$m_{b'} = m_{y_l}$

Fig. 3 Diagram of OT construction for AND gate

3.1 Our Construction

Let the function f to be evaluated between Alice and Bob with their private inputs $x = (x_1, x_2, \ldots, x_{v_1}) \in \{0,1\}^{v_1}$ and $y = (y_1, y_2, \ldots, y_{v_2}) \in \{0,1\}^{v_2}$ respectively where $v_1, v_2 \in \mathbb{N}$. Suppose that C be the acyclic Boolean circuit satisfying $C(x,y) = f(x,y)\ \forall x \in \{0,1\}^{v_1}, y \in \{0,1\}^{v_2}$. Moreover, assume that C is designed using AND and NOT gates. Then, Alice and Bob jointly execute input AND gates, input NOT gates, non-input AND gates and non-input NOT gates in the following manner.

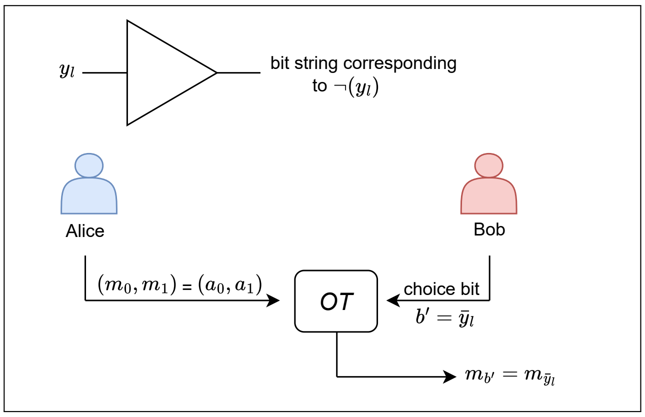
Input AND gate computation for Alice's input bit x_k and Bob's input bit y_l. Alice first chooses $a_0, a_1 \in_R \{0,1\}^\lambda$ associated with 0 and 1 respectively. Then Alice with inputs $(a_0, \bar{x}_k a_0 \oplus x_k a_1)$ and Bob with input choice bit y_l involve in a 1-out-of-2 OT. At the end of OT, Bob receives a_1 if $x_k = 1$ and $y_l = 1$, else it receives a_0. A schematic diagram of this construction is shown in Figure 3.

Input NOT gate computation for Bob's input y_l. In this case, Alice chooses $a_0, a_1 \in_R \{0,1\}^\lambda$ associated with 0 and 1 respectively. In the following, Alice with input (a_0, a_1) and Bob with input choice bit $\bar{y}_l$ involves in a 1-out-of-2 OT protocol, where $a_0, a_1 \in_R \{0,1\}^\lambda$ are associated with 0 and 1 respectively. On completion of OT protocol, Bob receives $a_{\bar{y}_l}$. A schematic diagram of this construction is shown in Figure 4.

Input NOT gate computation for Alice's input x_k. Alice chooses $a_{\bar{x}_k} \in_R \{0,1\}^\lambda$ and sends it to Bob, where a_0, a_1 are associated with 0 and 1 respectively.

Non-input AND gate computation for inputs $w_i, w_j \in \{0,1\}^\lambda$. Let $w_i \in \{a_0, a_1\}$ and $w_j \in \{b_0, b_1\}$ where a_0, b_0 are associated with 0's and a_1, b_1 are associated with 1's. Then Alice does the following,

(i) Alice chooses $c_0, c_1 \in_R \{0,1\}^\lambda$ corresponding to 0 and 1 respectively for the output wire.



y_l
bit string corresponding to $\neg(y_l)$
Alice
Bob
$(m_0, m_1) = (a_0, a_1)$
OT
choice bit $b' = \bar{y}_l$
$m_{b'} = m_{\bar{y}_l}$

Fig. 4 Diagram of OT construction for NOT gate

(ii) Makes a set consisting the following four entries

$$
\begin{aligned}
e_{00} &= a_0 \oplus b_0 \oplus c_0;\\
e_{01} &= a_0 \oplus b_1 \oplus c_0;\\
e_{10} &= a_1 \oplus b_0 \oplus c_0;\\
e_{11} &= a_1 \oplus b_1 \oplus c_1;
\end{aligned}
$$

(iii) Sends the set $E_{\mathsf{AND}} = \{e_{00}, e_{01}, e_{10}, e_{11}\}$ to Bob in a randomized order.
(iv) Sends $\{c_0, c_1\}$ in randomized order to Bob.

On receiving E_{AND} in some randomized order Bob does XOR of $w_i \oplus w_j$ with each member of the received set and checks whether it gets $c_\sigma \in \{c_0, c_1\}$ in one of the four cases. If not, then Bob outputs $\perp$ and aborts, else it sets the output $w_r = c_\sigma$. Note that, due to the XOR-ing, Bob will get either c_0 or c_1 in one of the four cases.

Non-input NOT gate computation for input $w_i \in \{0,1\}^\lambda$. Let $w_i \in \{a_0, a_1\}$ where a_0 is associated with 0, and a_1 is associated with 1. Then Alice does the following,

(i) Selects $c_0, c_1 \in_R \{0,1\}^\lambda$ associated with 0 and 1 respectively.
(ii) Makes a set containing

$$
\begin{aligned}
e_0 &= a_0 \oplus c_1;\\
e_1 &= a_1 \oplus c_0;
\end{aligned}
$$

(iii) Sends the set $E_{\mathsf{NOT}} = \{e_0, e_1\}$ to Bob in randomized order.
(iv) Forwards $\{c_0, c_1\}$ to Bob in some random order.

Bob on receiving E_{NOT} in randomized order, he does XOR of w_i with each elements of the received set and verifies whether it obtains $c_\sigma \in \{c_0, c_1\}$. If not, then outputs $\perp$

and abort, else writes the output $w_r = c_\sigma$. This is due to the fact that the XOR-ing will enable Bob to get either c_0 or c_1 in one of the two cases.

After executing all the input and non input gates, Bob remains with a string c_σ. In the following, Alice sends the set $\{(c_0, 0), (c_1, 1)\}$ to Bob for the final output gate of the circuit. As a consequence, Bob will be able to obtain the associated bit value of c_σ. In other words, Bob obtains the output of function $f(x, y)$ since $C(x, y) = f(x, y)$[1]. At the end, Bob will send $f(x, y)$ to Alice.

3.2 Correctness

In the following, the correctness of BooleanEval is discussed. We construct 1-out-of-2 OT for input gates. The correctness of 1-out-of-2 OT for AND and NOT gates is described below.

Correctness of 1-out-of-2 OT for input AND gate. The output of AND is 1, only when, both the input wire's value are 1; otherwise, it is 0. Recall that a_0, a_1 are bit strings associated with bit values 0 and 1, respectively. The construction of 1-out-of-2 OT for AND gate gives an output a_0 to Bob if the choice bit of Bob is $y_l = 0$. Therefore, when the input wire's value for AND gate is one of the pairs of $(x_k = 0, y_l = 0), (x_k = 1, y_l = 0)$, Bob obtains a_0, which is associated with the bit 0. On the other hand, Bob obtains $\bar{x}_k a_0 \oplus x_k a_1$ if the choice bit of Bob is $y_l = 1$. Therefore, when input wire's value for AND gate is the pair $(x_k = 0, y_l = 1)$, Bob obtains a_0 (putting $x_k = 0$). If the input wire's value for AND gate is the pair $(x_k = 1, y_l = 1)$, Bob obtains a_1 (putting $x_k = 1$), where a_1 is the bit string corresponding to the bit value 1. Therefore, the output of 1-out-of-2 OT correctly follows the truth table of AND gate.

Correctness of 1-out-of-2 OT for input NOT gate. The truth table of NOT gate shows that the output wire's value is 1 when the input wire's value is 0 and vice versa. It follows from the construction of 1-out-of-2 OT for NOT gate that Bob receives a_1 if his choice bit is $\bar{y}_l = 1$. That is, if the input wire's value of Bob is $y_l = 0$, he obtains bit string a_1 which is associated with bit 1. On the other hand, Bob obtains a_0 if the choice bit of Bob is $\bar{y}_l = 0$, i.e, if the input wire's value is Bob's is $y_l = 1$, he obtains the bit string a_1 corresponding to bit 1. Therefore, the output of 1-out-of-2 OT correctly follows the truth table of NOT gate.

Correctness of non input AND gate. Corresponding to each non-input AND gate, Bob has bit strings w_i, w_j corresponding to the input wires . Alice sends the set $E_{\mathsf{AND}} = \{e_{00}, e_{01}, e_{10}, e_{11}\}$ to Bob. Alice also sends the set $\{c_0, c_1\}$ associated with the output wire of the AND gate to Bob. Therefore, Bob has $w_i, w_j, E_{\mathsf{AND}}$ and $\{c_0, c_1\}$.

[1]Note that, if Alice wants to obtain the output of the function $f(x, y)$ by herself, she don't send the set $\{(c_0, 0), (c_1, 1)\}$ to Bob. Bob will send $w_r (= c_\sigma)$ to Alice and observing the value of c_σ, Alice will get the associated bit value, i.e., $f(x, y)$.

As per the construction 3.1

$$\begin{aligned} e_{00} &= a_0 \oplus b_0 \oplus c_0;\\ e_{01} &= a_0 \oplus b_1 \oplus c_0;\\ e_{10} &= a_1 \oplus b_0 \oplus c_0;\\ e_{11} &= a_1 \oplus b_1 \oplus c_1; \end{aligned}$$

where a_0, b_0 are random bit strings corresponding to 0 and a_1, b_1 are random bit strings corresponding to 1, associated with input wires of the AND gate. Clearly, $w_i \in \{a_0, a_1\}$ and $w_j \in \{b_0, b_1\}$. c_0, c_1 are randomly chosen bit strings by Alice corresponding to the output wire of that AND gate. It is easy to see that XOR-ing w_i, w_j with the set E_{AND} sent by Alice will enable Bob to obtain the correct output string c_σ for some $\sigma \in \{0, 1\}$.

$$\begin{aligned} e_{00} \oplus w_i \oplus w_j &= a_0 \oplus b_0 \oplus c_0 \oplus w_i \oplus w_j;\\ e_{01} \oplus w_i \oplus w_j &= a_0 \oplus b_1 \oplus c_0 \oplus w_i \oplus w_j;\\ e_{10} \oplus w_i \oplus w_j &= a_1 \oplus b_0 \oplus c_0 \oplus w_i \oplus w_j;\\ e_{11} \oplus w_i \oplus w_j &= a_1 \oplus b_1 \oplus c_1 \oplus w_i \oplus w_j; \end{aligned}$$

Since, $w_i \in \{a_0, a_1\}$ and $w_j \in \{b_0, b_1\}$, we can see that Bob obtains $c_\sigma = c_{i \cdot j}$ correctly as per the truth table of AND gate. In particular, if $w_i = a_0, w_j = b_1$, then from second expression it follows that Bob obtains $c_\sigma = c_0 = c_{\{0 \cdot 1\}}$ correctly. Similarly, if $w_i = a_0, w_j = b_0$, Bob obtains $c_\sigma = c_0 = c_{\{0 \cdot 0\}}$. For $w_i = a_1, w_j = b_0$, Bob obtains $c_\sigma = c_0 = c_{\{1 \cdot 0\}}$, and when $w_i = a_1, w_j = b_1$, from last expression, Bob obtains $c_\sigma = c_1 = c_{\{1 \cdot 1\}}$. Therefore, Bob correctly obtains a bit string c_σ associated with the output wire as per the truth table for AND gate.
Correctness of non-input NOT gate. Corresponding to each non-input NOT gate, Bob has bit strings w_i corresponding to the input wires. Alice sends the set $E_{\mathsf{NOT}} = \{e_0, e_1\}$ to Bob. Alice also sends $\{c_0, c_1\}$ associated with output wire of the NOT gate to Bob. Therefore, Bob has w_i, E_{NOT}, and $\{c_0, c_1\}$. As per the construction 3.1

$$\begin{aligned} e_0 &= a_0 \oplus c_1;\\ e_1 &= a_1 \oplus c_0; \end{aligned}$$

where a_0 is a random bit string corresponding to 0, and a_1 be the random bit strings corresponding to 1 associated with input wire of the NOT gate. Clearly, $w_i \in \{a_0, a_1\}$. c_0, c_1 are randomly chosen bit string by Alice corresponding to the output wire of that NOT gate. It is easy to see that XOR-ing w_i with E_{NOT} sent by Alice will enable Bob to obtain the correct output string c_σ for some $\sigma \in \{0, 1\}$

$$\begin{aligned} e_0 \oplus w_i &= a_0 \oplus c_1 \oplus w_i;\\ e_1 \oplus w_i &= a_1 \oplus c_0 \oplus w_i; \end{aligned}$$

Since, $w_i \in \{a_0, a_1\}$, we can see from the above data that Bob obtains $c_\sigma = c_{\neg i}$ correctly as per the truth table of NOT gate. In particular, if $w_i = a_0$, Bob correctly obtains $c_\sigma = c_1 = c_{\{\neg 0\}}$. If $w_i = a_1$, Bob correctly obtains $c_\sigma = c_0 = c_{\{\neg 1\}}$. Therefore, Bob correctly obtains the bit string c_σ associated with the output wire as per the truth table for NOT gate.

Thus, Bob can correctly compute all the non-input gates in topological order. As a consequence, Bob obtains the bit string c_σ corresponding to $C(x, y)$. At the end of the execution of all the gates in the circuit, Alice sends $\{(c_0, 0), (c_1, 1)\}$ corresponding to the final output gate to Bob. Clearly, $c_\sigma \in \{c_0, c_1\}$ as Bob correctly computed all the gates of the circuit. Matching the value of c_σ with $\{c_0, c_1\}$, Bob gets the output wire's bit string value of circuit C and associated bit value of that bit string correctly.

3.3 A Toy Example

In this section, we provide a toy example of BooleanEval. Let us consider a Boolean function $f : \{0,1\}^4 \times \{0,1\}^4 \to \{0,1\}$ defined by $f(x, y) = [\{(\neg(x_1 \cdot y_1)) + (\neg(x_2 \cdot y_2))\} \cdot (x_3 \cdot y_3)] \cdot \{\neg(x_4 + y_4)\}$. Using De Morgan's law we have,

$$\begin{aligned} f(x, y) &= [\{(\neg(x_1 \cdot y_1)) + (\neg(x_2 \cdot y_2))\} \cdot (x_3 \cdot y_3)] \cdot \{\neg(x_4 + y_4)\} \\ &= [\{\neg((x_1 \cdot y_1) \cdot (x_2 \cdot y_2))\} \cdot (x_3 \cdot y_3)] \cdot \{(\neg x_4) \cdot (\neg y_4)\} \qquad [\ by\ 1, 2] \end{aligned}$$

Alice has private input $x = 0101$ and Bob has private input $y = 1100$. Here, $v_1 = 4, v_2 = 4, u = 1$ and $x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 1; y_1 = 1, y_2 = 1, y_3 = 0, y_4 = 0$. Let λ be a security parameter. The Boolean circuit C corresponding to $f(x, y)$ is given in Figure 5. Let W_i be the output wire of G_i for $i = 1, 2, \ldots, 10$. We denote by w_i the bit string obtained by Bob corresponding to the output wire W_i of gate G_i for $i = 1, 2, \ldots, 10$. The computation of circuit is described below.

Computation of Input Gates ($G_1, \ldots, G_5$):

- Alice has private inputs $x_1 = 0, x_2 = 1, x_3 = 0$ and Bob has private inputs $y_1 = 1, y_2 = 1, y_3 = 0$ for G_1, G_2, G_3 (AND gates) respectively. Bob has private input $y_4 = 0$ for G_5 (NOT gate), and Alice has private inputs $x_4 = 1$ for G_4 NOT gate.
- To compute input AND gate G_1, Alice first chooses $a_0^1, a_1^1 \in_R \{0,1\}^\lambda$ associated with 0 and 1 respectively. In the following, Alice and Bob are involved in an 1-out-of-2 OT. Since $x_1 = 0$, Alice's input messages to OT are $(a_0^1, \bar{x}_1 a_0^1 \oplus x_1 a_1^1) = (a_0^1, a_0^1)$ and Bob's choice bit is $y_1 = 0$. Thus, Bob receives a_0^1 as an output of OT. Therefore $w_1 = a_0^1$.
  Similarly, to compute gate G_2, G_3, Alice first chooses a_0^2, a_1^2 and $a_0^3, a_1^3 \in_R \{0,1\}^\lambda$ where a_0^2, a_0^3 are associated with 0, and a_1^2, a_1^3 are associated with 1. Since $x_2 = 1$, Alice and Bob inputs to the OT are $(a_0^2, \bar{x}_2 a_0^2 \oplus x_2 a_1^2) = (a_0^2, a_1^2)$ and $y_2 = 1$ respectively. At the end, Bob obtains a_1^2 i.e., $w_2 = a_1^2$ as the output of OT. In a similar way, Bob obtains $w_3 = a_0^3$ as $x_3 = 0, y_3 = 0$ after computing the gate G_3.
- G_4 is an input NOT with Alice's input $x_4 = 1$. Alice chooses $a_0^4, a_1^4 \in_R \{0,1\}^\lambda$ associated with 0 and 1 respectively. Alice sends $a_{\bar{x}_4}^4 = a_0^4$ to Bob. Therefore, $w_4 = a_0^4$.

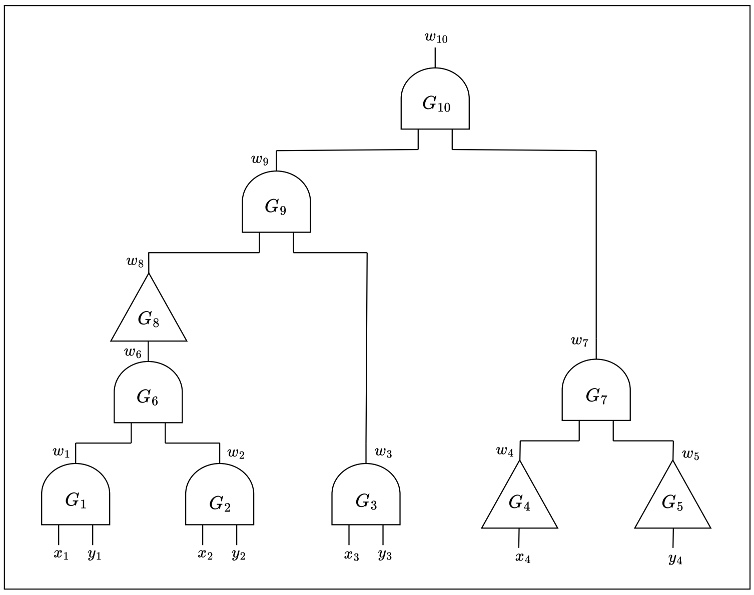


Fig. 5 Flow Diagram of Corresponding Boolean Circuit of Function f

- G_5 is an input NOT gate in which Bob's input is $y_5 = 0$. Alice first chooses $a_0^5, a_1^5 \in_R \{0,1\}^\lambda$ associated 0 and 1 respectively. Then Alice and Bob are involved in 1-out-of-2 OT with Alice's input (a_0^5, a_1^5) and Bob's choice bit $\bar{y}_5 = 1$. On completion of OT, Bob receives $w_5 = a_{\bar{y}_5}^5 = a_1^5$.

<u>**Computation of Non-input Gates** $(G_6 \dots G_{10})$**:**</u>

- Note that W_1, W_2 are input wires of AND gate G_6 and $w_1 \in \{a_0^1, a_1^1\}, w_2 \in \{a_0^2, a_1^2\}$. To compute G_6, Alice does the following,

(i) Alice chooses $a_0^6, a_1^6 \in_R \{0,1\}^\lambda$ corresponding to 0 and 1 respectively for the output wire W_6.

(ii) Makes a set with the following four entries

$$\begin{aligned} e_{00}^6 &= a_0^1 \oplus a_0^2 \oplus a_0^6; \\ e_{01}^6 &= a_0^1 \oplus a_1^2 \oplus a_0^6; \\ e_{10}^6 &= a_1^1 \oplus a_0^2 \oplus a_0^6; \\ e_{11}^6 &= a_1^1 \oplus a_1^2 \oplus a_1^6; \end{aligned}$$

(iii) Sends the set $E_{G_6} = \{e_{00}^6, e_{11}^6, e_{10}^6, e_{01}^6\}$, along with $\{a_0^6, a_1^6\}$ to Bob.

On receiving E_{G_6}, Bob does the following,

$$\begin{aligned} e_{00}^6 \oplus w_1 \oplus w_2 &= a_0^1 \oplus a_0^2 \oplus a_0^6 \oplus a_0^1 \oplus a_1^2 = \perp; \\ e_{01}^6 \oplus w_1 \oplus w_2 &= a_0^1 \oplus a_1^2 \oplus a_0^6 \oplus a_0^1 \oplus a_1^2 = a_0^6; \end{aligned}$$

$$e_{10}^6 \oplus w_1 \oplus w_2 = a_1^1 \oplus a_0^2 \oplus a_0^6 \oplus a_0^1 \oplus a_1^2 = \perp;$$
$$e_{11}^6 \oplus w_1 \oplus w_2 = a_1^1 \oplus a_1^2 \oplus a_1^6 \oplus a_0^1 \oplus a_1^2 = \perp;$$

Bob compares the output of the above XOR computations with the set $\{a_0^6, a_1^6\}$ to obtain $w_6 = a_0^6$.

- The AND gate G_7 has input wires W_4, W_5 and $w_4 \in \{a_0^4, a_1^4\}, w_5 \in \{a_0^5, a_1^5\}$. Similar to the computation of G_6, to compute G_7, Alice does the following,

(i) Alice chooses $a_0^7, a_1^7 \in_R \{0,1\}^\lambda$ corresponding to 0 and 1 respectively for the output wire W_7.
(ii) Makes a set with the following four entries

$$e_{00}^7 = a_0^4 \oplus a_0^5 \oplus a_0^7;$$
$$e_{01}^7 = a_0^4 \oplus a_1^5 \oplus a_0^7;$$
$$e_{10}^7 = a_1^4 \oplus a_0^5 \oplus a_0^7;$$
$$e_{11}^7 = a_1^4 \oplus a_1^5 \oplus a_1^7;$$

(iii) Sends $E_{G_7} = \{e_{00}^7, e_{01}^7, e_{11}^7, e_{10}^7\}$ along with $\{a_1^7, a_0^7\}$ to Bob.

On receiving E_{G_7}, Bob does the following,

$$e_{00}^7 \oplus w_4 \oplus w_5 = a_0^4 \oplus a_0^5 \oplus a_0^7 \oplus a_0^4 \oplus a_1^5 = \perp;$$
$$e_{01}^7 \oplus w_4 \oplus w_5 = a_0^4 \oplus a_1^5 \oplus a_0^7 \oplus a_0^4 \oplus a_1^5 = a_0^7;$$
$$e_{10}^7 \oplus w_4 \oplus w_5 = a_1^4 \oplus a_0^5 \oplus a_0^7 \oplus a_0^4 \oplus a_1^5 = \perp;$$
$$e_{11}^7 \oplus w_4 \oplus w_5 = a_1^4 \oplus a_1^5 \oplus a_1^7 \oplus a_0^4 \oplus a_1^5 = \perp;$$

Therefore, Bob obtains $w_7 = a_0^7$.

- The NOT gate G_8 has input wire W_6 and note that $w_6 \in \{a_0^6, a_1^6\}$. To compute G_8, Alice does the following,

(i) Alice selects $a_0^8, a_1^8 \in_R \{0,1\}^\lambda$ associated with 0 and 1 respectively for the output wire W_8.
(ii) Makes a set with following two entries

$$e_0^8 = a_0^6 \oplus a_1^8;$$
$$e_1^8 = a_1^6 \oplus a_0^8;$$

(iii) Sends the set $E_{G_8} = \{e_1^8, e_0^8\}$ along with $\{a_0^8, a_1^8\}$ to Bob.

On receiving E_{G_8}, Bob computes

$$e_0^8 \oplus w_6 = a_0^6 \oplus a_1^8 \oplus a_0^6 = a_1^8;$$
$$e_1^8 \oplus w_6 = a_1^6 \oplus a_0^8 \oplus a_0^6 = \perp;$$

Therefore, Bob obtains $w_8 = a_1^8$ after comparing the outputs of above XOR computations with the set $\{a_0^8, a_1^8\}$.

- Observe that W_8, W_3 are input wires of AND gate G_9, $w_8 \in \{a_0^8, a_1^8\}, w_3 \in \{a_0^3, a_1^3\}$. To compute G_9, Alice performs the following steps.

  (i) She chooses $a_0^9, a_1^9 \in_R \{0,1\}^\lambda$ corresponding to 0 and 1 respectively for the output wire W_9.

  (ii) Alice makes a set with the following four entries

$$\begin{aligned} e_{00}^9 &= a_0^8 \oplus a_0^3 \oplus a_0^9; \\ e_{01}^9 &= a_0^8 \oplus a_1^3 \oplus a_0^9; \\ e_{10}^9 &= a_1^8 \oplus a_0^3 \oplus a_0^9; \\ e_{11}^9 &= a_1^8 \oplus a_1^3 \oplus a_1^9; \end{aligned}$$

  (iii) Sends the set $E_{G_9} = \{e_{11}^9, e_{10}^9, e_{01}^9, e_{00}^9\}$ and $\{a_0^9, a_1^9\}$ to Bob.

  Bob performs the following XOR operations,

$$\begin{aligned} e_{00}^9 \oplus w_8 \oplus w_3 &= a_0^8 \oplus a_0^3 \oplus a_0^9 \oplus a_1^8 \oplus a_0^3 = \perp; \\ e_{01}^9 \oplus w_8 \oplus w_3 &= a_0^8 \oplus a_1^3 \oplus a_0^9 \oplus a_1^8 \oplus a_0^3 = \perp; \\ e_{10}^9 \oplus w_8 \oplus w_3 &= a_1^8 \oplus a_0^3 \oplus a_0^9 \oplus a_1^8 \oplus a_0^3 = a_0^9; \\ e_{11}^9 \oplus w_8 \oplus w_3 &= a_1^8 \oplus a_1^3 \oplus a_1^9 \oplus a_1^8 \oplus a_0^3 = \perp; \end{aligned}$$

  Hence, Bob obtains $w_9 = a_0^9$ after comparing the output of above XOR computations with the set $\{a_0^9, a_1^9\}$.

- G_{10} is the output gate of circuit C with input wires W_9 and W_7. Note that $w_9 \in \{a_0^9, a_1^9\}, w_7 \in \{a_0^7, a_1^7\}$. To compute G_{10}, Alice does the following,

  (i) Selects $a_0^{10}, a_1^{10} \in_R \{0,1\}^\lambda$ corresponding to 0 and 1 respectively for the output wire W_{10}.

  (ii) Makes a set with the following four entries

$$\begin{aligned} e_{00}^{10} &= a_0^9 \oplus a_0^7 \oplus a_0^{10}; \\ e_{01}^{10} &= a_0^9 \oplus a_1^7 \oplus a_0^{10}; \\ e_{10}^{10} &= a_1^9 \oplus a_0^7 \oplus a_0^{10}; \\ e_{11}^{10} &= a_1^9 \oplus a_1^7 \oplus a_1^{10}; \end{aligned}$$

  (iii) Sends the set $E_{G_{10}} = \{e_{00}^{10}, e_{11}^{10}, e_{10}^{10}, e_{01}^{10}\}$ and $\{(a_1^{10}, 1), (a_0^{10}, 0)\}$ to Bob.

  Bob receives $E_{G_{10}}$ and does the following XOR computations,

$$\begin{aligned} e_{00}^{10} \oplus w_9 \oplus w_7 &= a_0^9 \oplus a_0^7 \oplus a_0^{10} \oplus a_0^9 \oplus a_0^7 = a_0^{10}; \\ e_{01}^{10} \oplus w_9 \oplus w_7 &= a_0^9 \oplus a_1^7 \oplus a_0^{10} \oplus a_0^9 \oplus a_0^7 = \perp; \end{aligned}$$

Table 1 List of Symbols

Symbol	Meaning
x	Private input of Alice
y	Private input of Bob
v_1	Cardinality of x
v_2	Cardinality of y
u	Cardinality of output of function f
λ	length of arbitrary bit string chosen by Alice, security parameter
k	Index of Alice's input
v	number of input gates
l	Index of Bob's input
N	Size of circuit
$\in_R$	Randomly chosen
w_r	Corresponding bit string of output wire
c_σ	Corresponding value of output wire, obtained by Bob
$\overset{c}{\equiv}$	Indistinguishable symbol

$$e^{10}_{10} \oplus w_9 \oplus w_7 = a^9_1 \oplus a^7_0 \oplus a^{10}_0 \oplus a^9_0 \oplus a^7_0 = \perp;$$
$$e^{10}_{11} \oplus w_9 \oplus w_7 = a^9_1 \oplus a^7_1 \oplus a^{10}_1 \oplus a^9_0 \oplus a^7_0 = \perp;$$

- Bob obtains $w_{10} = a^{10}_0$ after comparing the outputs of the above XOR computations with the set $\{(a^{10}_1, 1), (a^{10}_0, 0)\}$ sent by Alice. Note that a^{10}_0 corresponds to the bit 0, thus Bob deduce that the output of the circuit C is 0. In the following, Bob sends 0 to Alice.

4 Security Analysis

In this section, we present the security analysis of BooleanEval. The security model of BooleanEval is presented below. A two-party secure Boolean Function Evaluation is a cryptographic protocol satisfying the following security properties.

- **Alice's privacy:** Bob obtains only the the output of the Boolean function, and can not obtain any information about the Alice's private inputs.
- **Bob's privacy:** Alice should not be able to obtain any information about Bob's private inputs.

To prove the security of the BooleanEval, we consider the following assumptions.

- The underlying OT protocol utilized to instantiate BooleanEval is secure.
- Alice and Bob behave in a semi-honest manner.

We will provide a simulation-based security proof of BooleanEval. We follow the approach of [2]. Intuitively, the protocol is secure if whatever is seen by its party can be computed only from that party's input and output. We denote the view of Alice to a protocol ϕ as V^ϕ_A and the view of Bob to protocol ϕ as V^ϕ_B. Consider two simulators S^1, S^2 that provide output for the view of Alice and Bob in the ideal world. We are going to show that $\{S^1(x, f(x, y))\} \overset{c}{\equiv} V^\phi_A$ and $\{S^2(y, f(x, y))\} \overset{c}{\equiv} V^\phi_B$.
Theorem 1. *Alice should be unable to obtain any information about Bob's private input.*

Proof. Let us assume that Alice is corrupted. Simulator S^1 is constructed in such a way that given input $(x, f(x,y))$, S^1 can simulate the view of Alice. Alice has private input x_k and assigns two random bit strings $a_0, a_1 \in_R \{0,1\}^\lambda$ to 0 and 1 respectively. Alice's view in this protocol consists only of the view in the OT protocol. Let S^1_{OT} denote the simulator for the secure 1-out-of-2 OT protocol. Alice's input for the OT protocol is $(a_0, a_1) \in_R \{0,1\}^\lambda$ (for input NOT gate) or $(a_0, \bar{x}_k a_0 \oplus x_k a_1)$ (for input AND gate). Since we assume that the OT protocol is secure, the simulator S^1 computes the OT protocol with Alice's inputs, and it is impossible to gather extra information about the choice bit, i.e., about the private input y_l of Bob. Therefore, it is easy to conclude that the output of S^1 is indistinguishable from the view of Alice, that is, $\{S^1(x, f(x,y))\} \overset{c}{\equiv} V^\phi_A$. □

Theorem 2. *Bob obtains only the output of the Boolean function and can not obtain any information about Alice's private inputs.*

Proof. We construct a simulator S^2 such that given input $(y, f(x,y))$ to S^2, it can simulate the view of Bob. Bob's view of this protocol consists of the OT protocol and the intermediate values obtained as the output of gates in the circuit. Let us consider simulator S^2_{OT} that simulates the OT protocol for Bob. The simulator S^2_{OT} takes input y_l and outputs $w_r = a_0$ or $a_1 \in_R \{0,1\}^\lambda$ (for NOT gate) and a_0 or $\bar{x}_k a_0 \oplus x_k a_1$ (for AND gate). However, S^2 does not know Alice's input x_k due to the security property of the OT protocol. Moreover, S^2 does not know the value E_G that Alice sends to Bob to compute the gate G, as S^2 only knows y and $f(x,y)$. Therefore, S^2 can not evaluate the circuit honestly. Next, S^2 will generate fake E_G and evaluate it as follows.

- S^2_{OT} has all the values corresponding to the output wires of input gates of the circuit C.
- S^2 sets two random bit strings for input wires of each non-input gate G. It chooses $a'_0, a'_1 \in_R \{0,1\}^\lambda$ associated to 1^{st} input wire and $b'_0, b'_1 \in_R \{0,1\}^\lambda$ associated to 2^{nd} input wire (in case of AND gate), and chooses $a"_0, a''_1 \in_R \{0,1\}^\lambda$ as an input wire's value (in case of non-input NOT gate). S^2 also sets a fake bit string $f'(x,y) \in \{0,1\}^\lambda$ corresponding to $f(x,y)$ and generates a fake set E_G consisting

$$e_{k_1 k_2} = \begin{cases} a'_{k_1} \oplus b'_{k_2} \oplus f'(x,y) & \text{if } G \text{ is } \mathsf{AND} \text{ gate} \end{cases}$$

$$e_{k_1} = \begin{cases} a''_{k_1} \oplus f'(x,y) & \text{if } G \text{ is } \mathsf{NOT} \text{ gate} \end{cases}$$

where $k_1, k_2 \in \{0,1\}$. S^2 runs the protocol and simulates the output $f(x,y)$. We will show that no polynomial time distinguisher D can distinguish the simulated and the real circuit evaluation output with non-negligible probability. We use the induction method to prove the above statement. We gate by gate in the topological order. Let σ_i include all the active secrets on the input wires of first i gates and let D_i be the corresponding polynomial time distinguisher. For the input gates G_i of the circuit C, the output wire's value w_i is obtained from the OT protocol, identically distributed in real and simulated cases. The OT protocol is secure in this model, and therefore, no distinguisher D_0 can obtain the corresponding values of the input or output wire's

Table 2 Comparison summary of SFE protocols

Protocol	Number of Commitments	Number of OT Executions	Number of Hash Evaluations	Number of XOR operations
Malkhi et al. [1]	mN	mN	$6mN$	$10mN$
Kolesnikov et al. [2]	-	v_2	$10N - 8\alpha$	$(10N - 5\alpha)(\lambda + 1) + 4N$
BooleanEval	-	v_2	-	$16\lambda(N-v)+v_2\lambda$

N = Number of logic gates, λ = Security parameter, m = Number of garbled circuits [1], α = Number of XOR gate in [2], v_2 = Number of private inputs of Bob, v = Number of input gates.

string w_i with non-negligible probability. Now let us assume that no polynomial time distinguisher D_{i-1} can distinguish transcripts σ_{i-1}, i.e., D_{i-1} can not obtain the secret values of input wires of the first $i-1$ gates with non-negligible probability. From the view of the i-th gate of the circuit C, S^2 only has values associated with input wires and some possible combinations of XOR values (E_{G_i}) of input wires. Moreover, D_{i-1} cannot conclude that the secret values correspond to input wires of $i-1$ gates with non-negligible probability. Thus, D_i cannot obtain any information about the input wires values of i-th gate. Therefore, there does not exist D_i that cannot distinguish the real and simulated transcript σ_i with non-negligible probability. Thus, S^2 is indistinguishable from the real view of Bob i.e., $\{S^2(y, f(x,y))\} \overset{c}{\equiv} V_B^{\phi}$. □

5 Efficiency and Comparison

We now present the efficiency analysis of BooleanEval. The proposed design utilizes 1-out-of-2 OT protocol as a building block to compute input gates. In addition, BooleanEval employs simple XOR operations to compute the final output of a circuit C. Let us assume that to construct the circuit C, N number of logic gates G_i are required. The complexity of BooleanEval majorly depends on the computation and communication complexity of the underlying OT used to instantiate BooleanEval. Let us assume that the underlying OT's communication and computation cost is β_1 and β_2, respectively. Let the number of input gates in the circuit be v.

Communication Complexity. A total of v_2 OT executions are required to evaluate a circuit C using BooleanEval. Thus, the total communication cost accrued during OT execution is $v_2\beta_1$. There are $v - v_2$ input NOT gates for Alice. λ bits are required to send the random bit strings a_i to Bob for each of the $v - v_2$ input NOT gates. Thus, a total of $(v - v_2)\lambda$ bits are required for this step. In addition, 4λ bits are needed to transmit E_{AND} for each non-input AND gate, while 2λ bits are required to transmit E_{NOT} for each non-input NOT gate. Moreover, 2λ bits are required to transmit the set $\{c_0, c_1\}$ for each of the non-input gates to Bob. Thus, at most 6λ bits are needed to be communicated for each non-input gate of the circuit C. For the computation of the final output, $2\lambda + 2$ bits are required to send (c_σ, σ) to Bob, and 1 bit is needed to send $f(x,y)$ to Alice. Thus, the total communication complexity of BooleanEval is $v_2\beta_1 + 6(N - v)\lambda + (v - v_2)\lambda + (2\lambda + 3)$.

Computation Complexity. Alice performs λ XOR operations for her input message $(a_0, \bar{x}_k a_0 \oplus x_k a_1)$ to the underlying OT protocol. Since a total of v_2 OT executions are required, the total cost incurred during OT execution is $v_2 \beta_2$. In constructing the set E_{AND} for each non-input AND gate, Alice needs to compute 8λ XOR operations. Similarly, to compute E_{AND} for each non-input AND gate, Bob needs to compute 8λ XOR operations. Similarly, in the case of each non-input NOT gate, Alice and Bob both require 2λ XOR operations. Since the size of the circuit C is N, a total of $16\lambda(N-v) + v_2\lambda$ XOR operations are needed to be performed. Therefore, the total computation cost of BooleanEval is $v_2\beta_2$ OT cost and $16\lambda(N-v)+v_2\lambda$ XOR operations.

We present a detailed comparative analysis of BooleanEval with the existing state-of-the-art protocols that employ similar design techniques, such as OT and XOR. In particular, we compare BooleanEval with [1] and [2]. The summary of our comparative analysis is provided in Table 2. As evident by the data in Table 2, BooleanEval is very efficient compared to [1, 2]. Unlike BooleanEval, [1] requires costly operations such as hash evaluations and commitments. In addition, number of OT executions in [1] is more than that in BooleanEval. The design of [2] also requires costly hash evaluations. On the design level, BooleanEval is better than [1, 2] as it only requires OT as a building block, whereas [1, 2] requires hash functions and commitment schemes as additional building blocks in their design. Although [2] does not require commitment schemes, it requires $10N - 8\alpha$ hash evaluations. Thus, overall BooleanEval performs better than [1, 2].

6 Conclusions

In this paper, we presented the design and security analysis of a two-party SBE protocol BooleanEval by using 1-out-of-2 OT as cryptographic building block. BooleanEval is efficient and fast as the only cost intensive operation employed in BooleanEval is XOR. A comparative analysis with the existing state-of-the-art protocols such as [1] and [2] showed the superiority of BooleanEval in terms of efficiency.

References

[1] Malkhi, D., Nisan, N., Pinkas, B., Sella, Y., *et al.*: Fairplay-secure two-party computation system. In: USENIX Security Symposium, vol. 4, p. 9 (2004). San Diego, CA, USA

[2] Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free xor gates and applications. In: Automata, Languages and Programming: 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II 35, pp. 486–498 (2008). Springer

[3] Cusick, T.W., Stanica, P.: Cryptographic Boolean Functions and Applications. Academic Press, (2017)

[4] Wu, C.-K., Feng, D.: Boolean Functions and Their Applications in Cryptography. Springer, (2016). https://doi.org/10.1007/978-3-662-48865-2 . http://dx.

doi.org/10.1007/978-3-662-48865-2

[5] Preneel, B.: Analysis and design of cryptographic hash functions. PhD thesis, Citeseer (1993)

[6] Schneider, T.: Practical secure function evaluation. In: Informatiktage, pp. 37–40 (2008)

[7] Blanton, M., Gasti, P.: Secure and efficient protocols for iris and fingerprint identification. In: Computer Security–ESORICS 2011: 16th European Symposium on Research in Computer Security, Leuven, Belgium, September 12-14, 2011. Proceedings 16, pp. 190–209 (2011). Springer

[8] Bringer, J., Chabanne, H., Favre, M., Patey, A., Schneider, T., Zohner, M.: Gshade: Faster privacy-preserving distance computation and biometric identification. In: Proceedings of the 2nd ACM Workshop on Information Hiding and Multimedia Security, pp. 187–198 (2014)

[9] Rouhani, B.D., Riazi, M.S., Koushanfar, F.: Deepsecure: Scalable provably-secure deep learning. In: Proceedings of the 55th Annual Design Automation Conference, pp. 1–6 (2018)

[10] Dowlin, N., Gilad-Bachrach, R., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: Cryptonets: applying neural networks to encrypted data with high throughput and accuracy. In: Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48. ICML'16, pp. 201–210. JMLR.org, (2016)

[11] Ramos-Casals, M., Brito-Zerón, P., Kostov, B., Sisó-Almirall, A., Bosch, X., Buss, D., Trilla, A., Stone, J.H., Khamashta, M.A., Shoenfeld, Y.: Google-driven search for big data in autoimmune geoepidemiology: analysis of 394,827 patients with systemic autoimmune diseases. Autoimmunity reviews **14**(8), 670–679 (2015)

[12] Riazi, M.S., Weinert, C., Tkachenko, O., Songhori, E.M., Schneider, T., Koushanfar, F.: Chameleon: A hybrid secure computation framework for machine learning applications. In: Proceedings of the 2018 on Asia Conference on Computer and Communications Security, pp. 707–721 (2018)

[13] Di Crescenzo, G., Feigenbaum, J., Gupta, D., Panagos, E., Perry, J., Wright, R.N.: Practical and privacy-preserving policy compliance for outsourced data. In: Financial Cryptography and Data Security: FC 2014 Workshops, BITCOIN and WAHC 2014, Christ Church, Barbados, March 7, 2014, Revised Selected Papers 18, pp. 181–194 (2014). Springer

[14] Yao, A.C.-C.: How to generate and exchange secrets. In: 27th Annual Symposium on Foundations of Computer Science (Sfcs 1986), pp. 162–167 (1986). IEEE

[15] Yao, A.C.: Protocols for secure computations. In: 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982), pp. 160–164 (1982). IEEE

[16] Shelat, A., Shen, C.-H.: Fast two-party secure computation with minimal assumptions. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, pp. 523–534 (2013)

[17] Jakobsson, M., Juels, A.: Mix and match: Secure function evaluation via ciphertexts. In: Advances in Cryptology—ASIACRYPT 2000: 6th International Conference on the Theory and Application of Cryptology and Information Security Kyoto, Japan, December 3–7, 2000 Proceedings 6, pp. 162–177 (2000). Springer

[18] Kamara, S., Mohassel, P., Riva, B.: Salus: a system for server-aided secure function evaluation. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, pp. 797–808 (2012)