An Efficient and Secure Boolean Function Evaluation Protocol

Sushmita Sarkar[1], Vikas Srivastava[2], Tapaswini Mohanty[3], Nibedita Kundu[4], Sumit Kumar Debnath[1], and Pantelimon Stănică[5]

[1] Department of Mathematics, National Institute of Technology Jamshedpur, Jamshedpur 831014, India
sarkarsushmita408@gmail.com, sdebnath.math@nitjsr.ac.in
[2] Department of Mathematics, National Institute of Technology Warangal, Telengana 506004, India
vikas.math123@gmail.com
[3] Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, Roorkee 247667, India
mtapaswini37@gmail.com
[4] Independent Researcher, West Bengal, India
nknkundu@gmail.com
[5] Applied Mathematics Department, Naval Postgraduate School, Monterey 93943, USA
pstanica@nps.edu

Abstract. The secure evaluation of Boolean functions or Secure Boolean Evaluation (SBE) is a fundamental component in this 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. The existing schemes need at least $(1.5\lambda+5)$-bit strings to garble an AND gate with security parameter λ, where λ is the length of arbitrary bit strings associated with inputs 0 and 1. In this manuscript, we discuss how we circumvent the communication cost of non-input gates independent of λ. 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. We introduce randomized Bloom filter (namely, RBF_S) associated with a set S to compute non-input AND gates. Moreover, BooleanEval only employs hash evaluations and XOR operations as the core computational step, making it lightweight and fast. Unlike other lightweight state-of-the-art designs of SBE (such as [16] and [18]), BooleanEval avoids the use of additional cryptographic primitives, such as 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. The secure evaluation of Boolean functions (also known as **S**ecure **B**oolean **E**valuation (SBE)) has wide-ranging applications in privacy-preserving protocols and secure multi-party computation. A two-party SBE allows two parties, Alice and Bob, to jointly evaluate a Boolean function without revealing their respective inputs. A two-party SBE protocol is called a two-way (or mutual SBE) protocol if both Alice and Bob can individually compute the desired output at the end of the execution of SBE. In contrast, a two-party SBE protocol is called a one-way SBE if only one of Alice and Bob can compute the desired output at the end of the SBE execution.

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) [23]. 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. SFE is employed to evaluate C securely.

SFE 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 [1,3], secure deep learning framework [22], privacy-preserving machine learning [7], and medical emergencies [19]. Besides, database mining and data storage outstanding [20,5] are some of the important applications of SBE. SFE is also a universal building block, and many interesting cryptographic protocols can be formulated as instances thereof, e.g., zero-knowledge proofs.

The first SFE protocol [26] was proposed by Yao as a solution to the famous millionaires' problem [25]. 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 SFE based on various techniques such as garbled circuit [26,16,14,24,21,18,27], mix and match [11], and server-aided [12].

In this work, we introduce a variant of the standard Bloom filter [2], namely randomized Bloom filter (RBF_S) associated with a set S. We focus on designing and analyzing a protocol for the secure evaluation of Boolean functions, in particular, a one-way SFE, using cryptographic techniques such as Oblivious Transfer (OT), hash evaluations, XOR operations, and RBF_S as building blocks.

Related Works. Malkhi et al. [16] 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 ones using the commitment scheme. At 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. [14] 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. Bob receives all the necessary information for computing the output. Bob evaluates the garbled circuit and obtains the output. However, these two protocols both require communication of 4-ciphertexts. In literature, the communication cost per AND gate is described by ciphertexts. However, it is possible to calculate each ciphertext in terms of λ, where λ is the length of bit strings associated with input 0 and 1 to the circuit C. Later, Pinkas et al. [18] introduced two optimization techniques and achieved the least communication cost of 2-ciphertexts. Zahur et al. [27] first reduce the data transfer into 2-ciphertexts for AND gates using the half gates concept. In that case, an AND gate is written as a combination of XORs and two half gates of opposite orientation. A half gate is a garbled AND gate for which one of the parties knows one of the inputs. Then, Rosulek et al. [21] applied a novel technique to the half gates model, which they called slicing and dicing, that involves slicing wire labels in half and operating separately on those halves. Each AND gate requires a communication cost of 1.5 ciphertexts. Therefore, the communication cost of AND gates includes the length of ar-

bitrary bit strings associated with inputs 0 and 1, for each of the existing well-known SFE protocols.

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 one-way SFE 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, such that Bob can 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 hash evaluations and XOR operations as the core computational step. These operations are efficient and fast. Thus, it leads to mini computational overhead and rapid execution.
- We introduce a variant of the standard Bloom filter, namely randomized Bloom filter RBF_S associated with a set S. RBF_S is designed to evaluate non-input AND gates efficiently, ensuring that the communication cost remains independent of the length (λ) of the bit strings associated with input values 0 and 1.
- BooleanEval does not require commitment scheme when compared to existing state-of-the-art lightweight designs such as [16,18]. The comparative analysis is discussed in the efficiency and comparison Section 5.

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

In this section, we present the preliminaries on Secure Function Evaluation, Bloom filters, and randomized Bloom filters. Definitions of Boolean functions, universal logic gates, and 1-out-of-2 oblivious transfer are provided in the Appendix A, B, and C.

2.1 Secure Function Evaluation

Secure Function Evaluation (SFE) [13] is a protocol that enables multiple parties to jointly compute the output of a function on their private inputs without revealing anything about those inputs beyond the function's output. In a two-party SFE protocol, if both parties individually can obtain the output, then the SFE is known as a two-way or mutual SFE protocol. If one of the parties obtains the output, then the SFE is known as a one-way SFE protocol.
Security Properties: In this manuscript, we provide the intuition of the security notions of SFE we use. We refer [13,15] for a formal definition and brief discussion.

- *Correctness.* Each party receives the correct output from the protocol.
- *Privacy.* No party should learn anything more than its prescribed output. In particular, the only information that should be learned about other parties' inputs is what can be derived from the output itself.

2.2 Bloom Filter [2]

A Bloom filter (BF) is a probabilistic data structure that represents a set $S = \{z_1, z_2, \ldots, z_n\}$ of n elements through an array of size m. It is parameterized with (m, n, k, H) and is denoted as BF_S. H be the set of k independent hash functions $\{h_0, h_1, \ldots, h_{k-1}\}$ with $h_\alpha : \{0,1\}^* \to \{0, 1, \ldots, m-1\}$ for each $\alpha = 0, 1, \ldots, k-1$. Each entry of BF_S is either 0 or 1. The element at i-th index of BF_S is denoted by $\mathsf{BF}_S[i]$. BF_S is used to represent a set S and to check the membership for an element in S. The Bloom filter BF_S comprises the following operations:

- *Initialization Phase:* Initialize an array of size m, setting each entry to 0.
- *Insertion Phase:* To insert an element $z \in S$ into BF_S, computes the hash values $h_0(z), h_1(z), \ldots, h_{k-1}(z)$. Then, set the entries at the corresponding positions $\mathsf{BF}_S[h_\alpha(z)]$ to 1 for each $\alpha = 0, 1, \ldots, k-1$. Repeat the process for each element $z \in S$.
- *Membership Test Phase:* To check the membership of an element $\overline{z}$ in the set S, one needs to compute $h_0(\overline{z}), h_1(\overline{z}), \ldots, h_{k-1}(\overline{z})$. Then check the elements $\mathsf{BF}_S[h_0(\overline{z})], \mathsf{BF}_S[h_1(\overline{z})], \ldots, \mathsf{BF}_S[h_{k-1}(\overline{z})]$. If $\mathsf{BF}_S[h_\alpha(\overline{z})] = 1;\ \forall \alpha = 0, 1, \ldots, k-1$, then $\overline{z}$ probably belongs to S. Otherwise, $\overline{z} \notin S$.

A Bloom filter attains false positive which means $\overline{z} \notin S$ falsely passes the membership test process, i.e., $\mathsf{BF}_S[h_\alpha(\overline{z})] = 1$ for all $\alpha = 0, 1, \ldots, k-1$.

Theorem 1. [6] *Given the number n of elements to be added and a desired maximum false positive rate $\frac{1}{2^k}$, the optimal size m of the Bloom filter is $m = \frac{nk}{\ln 2}$.*

2.3 Randomized Bloom Filter

We now introduce a variant of the standard Bloom filter, namely randomized Bloom filter (RBF), where the entries are filled with bit strings of some fixed length $\eta \in \mathbb{N}$ instead of 0 and 1. We denote the randomized Bloom filter corresponding to a set S as RBF_S and it is parameterized with (n, m, k, H, η). The element at i-th index of RBF_S is denoted by $\mathsf{RBF}_S[i]$. The construction of RBF_S is as follows:

- *Initialization Phase:* Initialize an array of size m with each position set to a null value ($\perp$), for instance, a string of zeros.
- *Insertion Phase:*
  1. To insert a value $r \in \{0,1\}^\eta$ corresponding to an element $z \in S$, Alice computes k hash values $h_0(z), h_1(z), \ldots, h_{k-1}(z)$ to find the insertion indices in the array.
  2. In the following, Alice proceeds as follows:
     (a) Finds all the indices $\alpha \in \{0, 1, \ldots k-1\}$ such that $\mathsf{RBF}_S[h_\alpha(z)] = \perp$ and reserves an index j from these indices for final share.
     (b) For other $\alpha (\neq j)$ with $\mathsf{RBF}_S[h_\alpha(z)] = \perp$, sets $\mathsf{RBF}_S[h_\alpha(z)] = t_\alpha \in_R \{0,1\}^\eta$.
     (c) Finally, sets $\mathsf{RBF}_S[h_j(z)] = r \overset{k-1}{\underset{\substack{\alpha=0 \\ \alpha \neq j}}{\oplus}} \mathsf{RBF}_S[h_\alpha(z)]$.

     Throughout this paper, we call the bit strings $t_\alpha (\alpha = 0, \ldots, k-1)$ as the shares of r.
  3. This process is repeated for each element $z \in S$.
  4. All the remaining entries in RBF_S are filled with a random bit string of length η.
- *Retrieve Phase:* To retrieve the value r corresponding to an element $z \in S$, Bob calculates the indices $h_0(z), h_1(z), \ldots, h_{k-1}(z)$ and computes the XOR sum of the values at those positions: $r = \overset{k-1}{\underset{\alpha=0}{\oplus}} \mathsf{RBF}_S[h_\alpha(z)]$.

Algorithm 1 BuildRBF(S, n, m, k, H, η)

Input: A set S, cardinality of S is n, array size m, number of hash functions k, length of the value η that to be inserted in RBF_S, hash family $H = \{h_0, \ldots, h_{k-1}\}$
Output: An (n, m, k, H, η)-randomized Bloom filter RBF_S

$\mathsf{RBF}_S \leftarrow$ new m-element array of bit strings;
for $i = 0$ to $m - 1$ **do**
    $\mathsf{RBF}_S[i] \leftarrow \perp$; // $\perp$ is the symbol meaning "a string of zeros"
end for
for each (z, r) corresponding to an element $z \in S$ **do**
    emptySlot $\leftarrow -1$;
    finalShare $\leftarrow r$;
    for $i = 0$ to $k - 1$ **do**
        $j \leftarrow h_i(z)$; // get an index by hashing the element
        if $\mathsf{RBF}_S[j] == \perp$ **then**
            if emptySlot$== -1$ **then**
                emptySlot $\leftarrow j$; // reserve this location for finalShare
            else
                $\mathsf{RBF}_S[j] \xleftarrow{R} \{0,1\}^\eta$; // generate a new share
                finalShare $\leftarrow$ finalShare $\oplus \mathsf{RBF}_S[j]$;
            end if
        else
            finalShare $\leftarrow$ finalShare $\oplus \mathsf{RBF}_S[j]$;
        end if
    end for
    RBF_S[emptySlot] $\leftarrow$ finalShare; // store the last share
end for
for $i = 0$ to $m - 1$ **do**
    if $\mathsf{RBF}_S[i] == \perp$ **then**
        $\mathsf{RBF}_S[i] \xleftarrow{R} \{0,1\}^\eta$;
    end if
end for

Algorithm 2 RetrieveRBF$(z, m, k, H, \mathsf{RBF}_S)$

Input: An element z, array size m, number of hash functions k, hash family $H = \{h_0, \ldots, h_{k-1}\}$, randomized Bloom filter RBF_S
Output: Retrieved value r

$r \leftarrow 0^\eta$
for $i = 0$ to $k - 1$ **do**
    $j \leftarrow h_i(z)$ // compute the hash index
    $r \leftarrow r \oplus \mathsf{RBF}_S[j]$
end for
return r

Note that, unlike the standard Bloom filter (BF_S), the randomized Bloom filter (RBF_S) does not allow to check the presence of an element z in S. Rather, it allows to retrieve an element r associated with $z \in S$ from RBF_S through the retrieve phase. The correctness of this retrieval relies on the linear property of XOR. When a value is retrieved, the XOR operation effectively "sums" the shares that were inserted. The false positive rate of the associate Bloom filter BF_S for an RBF_S determines the probability that an element $z \notin S$ maps to the positions which are already filled with the shares for some members in S. The algorithm to insert a value into a randomized Bloom filter and the algorithm to retrieve a value corresponding to an element belonging to the set are given in Algorithms 1 and 2.

3 Proposed Secure Boolean Function Evaluation Protocol BooleanEval

A high level overview: In this work, we focus on the design of a two-party secure one-way Boolean function evaluation protocol, namely 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 $v_1, v_2 \in \mathbb{N}$. The objective is such that Bob can 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 [14] 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 are known as universal logic gates. Assume that the circuit C has M number of non-input NOT gates and N number of non-input AND gates.

System parameters. Let RBF_S be a randomized Bloom filter, associated with a set S, which is parameterized by (m, n, k, H, η) (refer Section 2.2). Consider H be the set of k independent hash functions $h_\alpha : \{0,1\}^* \longrightarrow \{0, \ldots, m-1\}$, where $\alpha \in \{0, 1, \ldots, k-1\}$. The entries of RBF_S are bit strings of length η. In this paper we utilize RBF_S to encode each element of the set $\{r^\gamma, r^\gamma, r^\gamma, s^\gamma\}_{\gamma=1}^N$ associated with the corresponding position's element of the set $S = \{z_0^\gamma, z_1^\gamma, z_2^\gamma, z_3^\gamma\}_{\gamma=1}^N$ of length $n = 4N$ in an array of optimal length m. The detailed construction of RBF_S for non-input AND gates is described later. Also consider h to be a cryptographically secure hash function defined as $h : \{0,1\}^\eta \longrightarrow \{0,1\}^\lambda$.

Note that, here λ is a security parameter, that is the length of arbitrary strings associated with input values 0 and 1. For a given security level, λ must be at least 256 bits to provide adequate security against modern cryptographic attacks. The parameter η determines the length of the strings (index values $\mathsf{RBF}_S[i]$) stored in the RBF_S. Basically, in BooleanEval, η is the length of shares r^γ, s^γ. Thus, η has no such security bounds, unlike λ. One can choose $\eta = 2$ or 3. Another parameter k is the number of independent hash functions. In our scheme, if the false positive probability of the corresponding BF_S is high, then insertion of a new member z in RBF_S may not be feasible with high probability, as all $h_i(z)$ may map to the position of previously occupied shares. Thus false positive rate of the corresponding BF_S should be negligible. Moreover, the false positive probability of the associated Bloom filter BF_S for an RBF_S is upper bounded by $\frac{1}{2^k}$ (refer 1). For $k = 20$, $\frac{1}{2^k} \approx 9.54 \times 10^{-7}$, which can be considered as a negligible probability. Therefore, to implement our protocol, $\eta = 2$ or 3 and $k = 20$ may be considered. While λ, k, η are parameters, they serve distinct purposes and do not necessarily scale linearly with each other. The communication independence from λ for non-input gates is a key advantage, as Bob only needs to receive the RBF_S, where the size depends on the length of k, η and the number of non-input AND gates, not on the length of the wire labels. The above-mentioned hash function (h) can be instantiated with SHA-2, SHA-3, or BLAKE2.

Protocol's high level description. The input AND gates and input NOT gates with respect to the input y_p of Bob are obliviously evaluated by executing an OT protocol between Alice and Bob. For each input NOT gate with respect to input x_q of Alice, an arbitrary bit string $a_{\bar{x}_q}$ is sent to Bob by Alice, where $a_0, a_1 \in_R \{0,1\}^\lambda$. In case of OT for input AND gates with respect to Alice's input x_q and Bob's input y_p, Alice's input pair to OT is of the form $(a_0, \bar{x}_q a_0 \oplus x_q a_1)$ and Bob's choice bit is y_p where $a_0, a_1 \in \{0,1\}^\lambda$. While in case of OT for input NOT gates with respect to input y_p of Bob, Alice's input pair is (a_0, a_1) and Bob's choice bit is $\bar{y}_p$, where $a_0, a_1 \in_R \{0,1\}^\lambda$ corresponding to bit 0 and 1 respectively.

For the case of non-input AND gates $G^\gamma_{\mathsf{AND}}(\gamma = 1, \ldots, N)$, Alice sets $S = \{z^\gamma_0, z^\gamma_1, z^\gamma_2, z^\gamma_3\}^N_{\gamma=1}$, where $z^\gamma = (a^\gamma_i || b^\gamma_j)$ according to the truth table of AND gate for inputs $a^\gamma_i, b^\gamma_j \in \{0,1\}^\lambda$ $(i, j \in \{0,1\})$. Alice then constructs a randomized Bloom filter RBF_S to insert $r^\gamma \in_R \{0,1\}^\eta$ corresponding to each element of $\{z^\gamma_0, z^\gamma_1, z^\gamma_2\}$, $s^\gamma \in_R \{0,1\}^\eta \setminus r^\gamma$ corresponding to $\{z^\gamma_3\}$ for $\gamma = 1, \ldots, N$. Moreover, other $2k$ positions are adjusted to insert shares of s^γ. The output of the AND gate G^γ_{AND} is set to $h(r^\gamma)$ or $h(s^\gamma)$ associated with 0 or 1 respectively. Finally, Alice sends RBF_S to Bob.

Now, corresponding to each non-input AND gate G^γ_{AND}, Bob can easily compute $w^\gamma_i || w^\gamma_j$ to get z^γ_l for some $l \in \{0, \ldots, 3\}$ with inputs $w^\gamma_i, w^\gamma_j \in \{0,1\}^\lambda$. Then, he calculates $\rho^\gamma = \bigoplus_{\alpha=0}^{k-1} \mathsf{RBF}_S[h_\alpha(z^\gamma_l)]$ and gets either r^γ or s^γ according to z^γ_l, he has computed. Thus, Bob gets the output of non-input AND gates as $h(\rho^\gamma)$ for $\gamma = 1, \ldots, N$.

For each non-input NOT gate $G^\gamma_{\mathsf{NOT}}(\gamma = 1, \ldots, M)$, Alice chooses $\delta^\gamma \in_R \{0,1\}^\eta$ and computes $\Delta^\gamma = h(\delta)^\gamma$. Let a^γ_0 or a^γ_1 be the input of non-input NOT gate G^γ_{NOT} associated with 0 or 1 respectively. She sets the output as $c^\gamma_0 = a^\gamma_1 \oplus \Delta^\gamma$ corresponding to 0 and $c^\gamma_1 = a^\gamma_0 \oplus \Delta^\gamma$ corresponding to 1. Then, Alice sends the set $\{\delta^\gamma\}^M_{\gamma=1}$ to Bob. In the following, Bob does $w^\gamma_i \oplus h(\delta^\gamma) = w^\gamma_i \oplus \Delta^\gamma$ for input $w^\gamma_i \in \{0,1\}^\lambda$ he has obtained for $\gamma = 1, \ldots, M$. Thus, he gets the output of the non-input NOT gates.

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.

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 $C(x, y) = f(x, y)$ $\forall x \in \{0,1\}^{v_1}, y \in \{0,1\}^{v_2}$. The circuit C is designed using AND and NOT gates. In addition, let there are N non-input AND gates and M non-input NOT gates. We denote the γ-th non-input AND gate as G^γ_{AND} and NOT gate as G^γ_{NOT}. We have introduced randomized Bloom filter (RBF_S) associated with a set S (refer Section 2.2) to design our protocol. In the following, we discuss how RBF_S is used to compute non-input AND gates. The randomized Bloom filter (namely, RBF_S) is parameterized with (m, n, k, H, η). Each entry in RBF_S is a bit string of length η. It encodes shares of length η corresponding to a set S of cardinality n into an array of optimal length m. $H = \{h_\alpha\}^{k-1}_{\alpha=0}$ be the set of k independent hash functions, where each hash function is defined as $h_\alpha : \{0,1\}^* \longrightarrow \{0, \ldots, m-1\}$ for $\alpha = 0, 1, \ldots, k-1$. Moreover, $h : \{0,1\}^\eta \longrightarrow \{0,1\}^\lambda$ be another cryptographically secure hash function, where λ is the length of arbitrary bit strings associated with the input bit values 0 and 1. Alice and Bob jointly execute input AND gates, input NOT gates, non-input AND gates and non-input NOT gates in the following:

Input AND gates computation for Alice's input bit x_q, Bob's input bit y_p. In the follow-

ing for each input AND gate, Alice first chooses $a_0, a_1 \in_R \{0,1\}^\lambda$ associated with input values 0 and 1 respectively. Then Alice with inputs $(a_0, \bar{x}_q a_0 \oplus x_q a_1)$ and Bob with input choice bit y_p are involved in a 1-out-of-2 OT. At the end of OT, Bob receives a_1 if $x_q = 1$ and $y_p = 1$, else it receives a_0. A schematic diagram of this construction is shown in Figure 1.

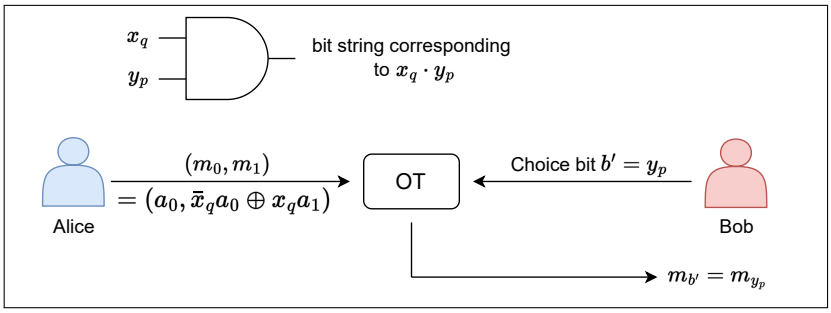


Fig. 1. Diagram of OT construction for AND gate

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

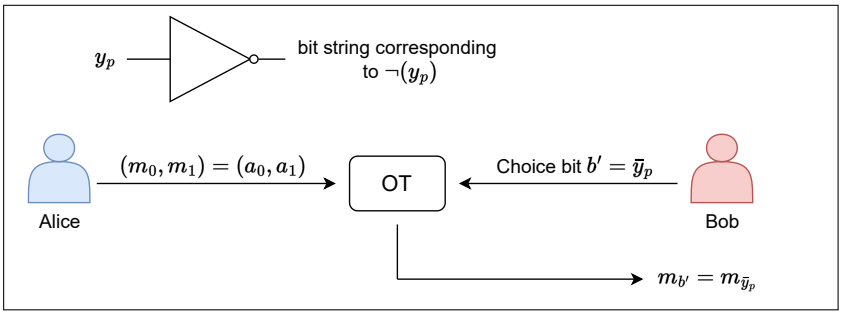


Fig. 2. Diagram of OT construction for NOT gate

Input NOT gates computation for Alice's input x_q. To compute this kind of input NOT gates, for each gate Alice chooses $a_{\bar{x}_q} \in_R \{0,1\}^\lambda$ and sends it to Bob, where a_0, a_1 are associated with input values 0 and 1 respectively.

Non-input AND gates computation. Let us assume that there are N number of non-input AND gates. Suppose, each of the γ-th non-input AND gate $G^{\gamma}_{\mathsf{AND}}$ have two input wires with inputs $w_i^{\gamma} \in \{a_0^{\gamma}, a_1^{\gamma}\}$, $w_j^{\gamma} \in \{b_0^{\gamma}, b_1^{\gamma}\}$, where $a_0^{\gamma}, b_0^{\gamma} \in \{0,1\}^{\lambda}$ are associated with 0's and $a_1^{\gamma}, b_1^{\gamma} \in \{0,1\}^{\lambda}$ are associated with 1's. *Alice does the following*:

(i) Computes

$$\begin{aligned} a_0^{\gamma}||b_0^{\gamma} &= z_0^{\gamma};\\ a_0^{\gamma}||b_1^{\gamma} &= z_1^{\gamma};\\ a_1^{\gamma}||b_0^{\gamma} &= z_2^{\gamma};\\ a_1^{\gamma}||b_1^{\gamma} &= z_3^{\gamma}. \end{aligned} \tag{1}$$

for each non-input AND gate $G^{\gamma}_{\mathsf{AND}}(\gamma = 1, \ldots, N)$. Writes the set $S = \{z_0^{\gamma}, z_1^{\gamma}, z_2^{\gamma}, z_3^{\gamma}\}_{\gamma=1}^{N}$. Thus, the value of one of the parameters of GBF is $n = 4N$.

(ii) Chooses two distinct random bit strings $r^{\gamma}, s^{\gamma} \in_R \{0,1\}^{\eta}$ (i.e., $r^{\gamma} \neq s^{\gamma}$) for each $\gamma = 1, \ldots, N$.

(iii) Sets the values $c_0^{\gamma} = h(r^{\gamma}), c_1^{\gamma} = h(s^{\gamma}) \in_R \{0,1\}^{\lambda}$ associated with 0's and 1's for the output wires respectively for $\gamma = 1, \ldots, N$[6].

(iv) Constructs a randomized Bloom filter (namely, RBF_S) for the set S as follows:

(a) Initializes an array of size m with all entries set to a null value ($\perp$), for instance, a string of zeros.

(b) For each $\gamma = 1, \ldots, N$, she inserts r^{γ} associated to the set $\{z_l^{\gamma}\}_{l=0}^{2}$. We denote the shares of r^{γ} as $\{t_{l\alpha}^{\gamma}\}_{\alpha=0}^{k-1}$ for each $l = 0, 1, 2$. Note that, the shares $\{t_{l\alpha}^{\gamma}\}$ are located at the corresponding positions $\{h_{\alpha}(z_l^{\gamma})\}$ (refer the RBF insertion phase 2.2).

(c) Similarly, for the value s^{γ} associated with z_3^{γ}, she inserts s^{γ} to RBF_S. We denote the shares of s^{γ} as $\{\widetilde{t}_{3\alpha}^{\gamma}\}_{\alpha=0}^{k-1}$. Note that, the shares $\{t_{3\alpha}^{\gamma}\}$ are located at the corresponding positions $\{h_{\alpha}(z_3^{\gamma})\}$ (refer the RBF insertion phase 2.2). That is,

$$\bigoplus_{\alpha=0}^{k-1} \mathsf{RBF}_S[h_{\alpha}(z_l^{\gamma})] = \begin{cases} r^{\gamma} & \text{if } l = 0, 1, 2\\ s^{\gamma} & \text{if } l = 3. \end{cases}$$

(d) Randomly selects $2N$ distinct indices

$$A_4 = \{\beta_1^1, \beta_2^1, \beta_1^2, \beta_2^2, \ldots, \beta_1^N, \beta_2^N\} \in \{0, 1, \ldots, m-1\} \setminus (A_0 \cup A_1 \cup A_2 \cup A_3),$$

where $A_l = \{h_0(z_l^{\gamma}), h_1(z_l^{\gamma}), \ldots, h_{k-1}(z_l^{\gamma})\}_{\gamma=1}^{N}$ with $l = 0, 1, 2, 3$[7].

[6] Note that, here we have used r^{γ} corresponds to first three combinations of Eq. (1) that is used to obtain the output associated with 0, and s^{γ} corresponds to the last combination of Eq. (1) that is used to obtain the output associated with 1. This correspondence helps us to reach the criterion of AND gate's truth table output column.

[7] Here, one may have issue to insert $\bar{r}$ in RBF_S for a new element $\bar{z} \in S$ in insertion phase, if all the indices $h_0(\bar{z}), h_1(\bar{z}), \ldots, h_{k-1}(\bar{z})$ are already occupied by bit strings of length η due to the already inserted r's corresponding to z's in S. However, the probability will appear with negligible probability if the false positive rate of the associated BF_S is negligible. To achieve $\frac{1}{2^k}$ as false positive rate of Bloom filter BF_S, the randomized Bloom filter (RBF_S) parameters are set as $m = \frac{kn}{\ln 2}$, where m is the size of RBF_S, k is the number of hash functions used in RBF_S, n is the cardinality of the set S, defined in step (i). If we choose $k \geq 20$ then $\frac{1}{2^k}$ becomes negligible probability as $\frac{1}{2^{20}} \approx 9.54 \times 10^{-7}$. Now $m = \frac{kn}{\ln 2} \approx 1.4kn = kn + 0.4kn$. Note that, to insert the set S of cardinality n into the RBF_S, we may require at most kn entries, i.e., $|A_0 \cup A_1 \cup A_2 \cup A_3| \leq kn$. So, there are at least $0.4kn \geq 8n = 32N$ many remaining entries in the RBF_S. Note that in step 4(e), we need only $2N$ entries. Thus, the optimal length m of RBF_S is sufficient to execute the protocol. We need not increase the optimal value of m further.

(e) For each $\gamma = 1, 2, \ldots, N$, $\mathsf{RBF}_S[\beta_1^\gamma]$ and $\mathsf{RBF}_S[\beta_2^\gamma]$ are filled with bit strings $s^\gamma \overset{k-2}{\underset{\alpha=0}{\oplus}} \mathsf{RBF}_S[\phi_\alpha^\gamma]$ and $s^\gamma \overset{k-2}{\underset{\alpha=0}{\oplus}} \mathsf{RBF}_S[\psi_\alpha^\gamma]$ for randomly chosen $2k-2$ indices $\{\phi_1^\gamma, \psi_1^\gamma, \phi_2^\gamma, \psi_2^\gamma, \ldots, \phi_{k-1}^\gamma, \psi_{k-1}^\gamma\}$ from $A_0 \cup A_1 \cup A_2 \cup A_3$.

(f) Each of the remaining indices $\mathsf{RBF}_S[\beta]$ is filled with randomly chosen bit strings of length η.

(v) Finally, Alice sends the RBF_S to Bob.

On receiving RBF_S *from Alice, Bob executes the following steps for each* $\gamma = 1, 2, \ldots, N$.

(i) Computes $(w_i^\gamma || w_j^\gamma)$ to get the corresponding $z_l^\gamma \in \{z_0^\gamma, z_1^\gamma, z_2^\gamma, z_3^\gamma\}$.

(ii) Evaluates $\overset{k-1}{\underset{\alpha=0}{\oplus}} \mathsf{RBF}_S[h_\alpha(z_l^\gamma)]$ to get $\rho^\gamma \in \{r^\gamma, s^\gamma\}$.

(iii) Computes the output of non-input AND gate G_{AND}^γ as $h(\rho^\gamma) \in \{c_0^\gamma, c_1^\gamma\}$.

Non-input NOT gates computation. Consider the circuit C with M number of non-input NOT gates $G_{\mathsf{NOT}}^\gamma (\gamma = 1, 2, \ldots, M)$. For each γ, let $w^\gamma \in \{a_0^\gamma, a_1^\gamma\}$ be the input wire of G_{NOT}^γ, where $a_0^\gamma, a_1^\gamma \in_R \{0,1\}^\lambda$ are associated with 0's, 1's respectively. *Alice does the following for each* $\gamma = 1, 2, \ldots, M$.

(i) Chooses $\delta^\gamma \in_R \{0,1\}^\eta$ and computes $\Delta^\gamma = h(\delta^\gamma)$ for non-input NOT gate G_{NOT}^γ.

(ii) Sets the output of G_{NOT}^γ as

$$\begin{aligned} c_0^\gamma &= a_1^\gamma \oplus \Delta^\gamma, \text{ corresponds to the bit 0} \\ c_1^\gamma &= a_0^\gamma \oplus \Delta^\gamma, \text{ corresponds to the bit 1.} \end{aligned} \tag{2}$$

(iii) Sends $\{\delta^\gamma\}_{\gamma=1}^M$ to Bob.

On receiving $\{\delta^\gamma\}_{\gamma=1}^M$, *Bob does the following for each* $G_{\mathsf{NOT}}^\gamma (\gamma = 1, 2, \ldots, M)$.

(i) Computes $\Delta^\gamma = h(\delta^\gamma)$ and $w^\gamma \oplus \Delta^\gamma$.

(ii) Sets the output of the G_{NOT}^γ as $w^\gamma \oplus \Delta^\gamma$.

Note that, due to Equation 2, Bob's output will be c_1^γ when $w^\gamma = a_0^\gamma$ and c_0^γ when $w^\gamma = a_1^\gamma$.

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 by comparing c_σ with c_0, c_1. In other words, Bob obtains the output of function $f(x, y)$ since $C(x, y) = f(x, y)$[8]. A toy example for the proposed design is provided in Appendix D.

[8] Note that, if Alice wants to obtain the output of the function $f(x, y)$ by herself, she doesn'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., $f(x, y)$.

3.2 Correctness

In the following, the correctness of BooleanEval is discussed. We construct 1-out-of-2 OT for input gates and randomized Bloom filter (RBF_S) for non-input AND gates. The correctness of AND and NOT gates for the circuit C is described below.

Correctness of 1-out-of-2 OT for input AND gate. The output of AND gate is 1, only when both the input wires' values 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 $y_p = 0$. Therefore, when the input wire's value for AND gate is one of the pairs of $(x_q = 0, y_p = 0)$ or $(x_q = 1, y_p = 0)$, Bob obtains a_0, which is associated with the bit 0. On the other hand, Bob obtains $\bar{x}_q a_0 \oplus x_q a_1$ if the choice bit of Bob is $y_p = 1$. Therefore, when input wire's value for AND gate is the pair $(x_q = 0, y_p = 1)$, Bob obtains a_0 (putting $x_q = 0$). If the input wire's value for AND gate is the pair $(x_q = 1, y_p = 1)$, Bob obtains a_1 (putting $x_q = 1$), 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 $\bar{y}_p = 1$. That is, if the input wire's value is $y_p = 0$, Bob 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 $\bar{y}_p = 0$. That is, if the input wire's value is $y_p = 1$, he obtains the bit string a_0 corresponding to bit 0. Therefore, the output of 1-out-of-2 OT correctly follows the truth table of NOT gate. However, for input NOT gate with Alice's input x_q, an arbitrary bit string $a_{\bar{x}_q}$, associated with $\bar{x}_q$, is sent to Bob. Thus, if Alice's input for NOT gate is $x_q = 0$, Bob obtains $a_{\bar{x}_q} = a_1$, corresponding to bit value 1. If Alice's input for NOT gate is $x_q = 1$, Bob obtains $a_{\bar{x}_q} = a_0$, corresponding to bit value 0. Therefore, the construction for input NOT gates with Alice's input to the wire follows NOT gate's truth table correctly.

Correctness of non input AND gate. Corresponding to each non-input AND gate $G^{\gamma}_{\mathsf{AND}}(\gamma = 1, \ldots, N)$, Bob has bit strings $w^{\gamma}_i, w^{\gamma}_j$ associated with the input wires. To compute $G^{\gamma}_{\mathsf{AND}}(\gamma = 1, \ldots, N)$, Alice sends RBF_S to Bob. According to our construction from Section 3.1, the output of non-input AND gates is set to be $c^{\gamma}_0 = h(r^{\gamma})$ associated with the bit value 0 or $c^{\gamma}_1 = h(s^{\gamma})$ associated with the bit value 1. Note that in RBF_S, r^{γ} corresponds to each element of $\{z^{\gamma}_0, z^{\gamma}_1, z^{\gamma}_2\}$ and s^{γ} corresponds to z^{γ}_3. That is,

$$\bigoplus_{\alpha=0}^{k-1} \mathsf{RBF}_S[h_\alpha(z^{\gamma}_0)] = \bigoplus_{\alpha=0}^{k-1} \mathsf{RBF}_S[h_\alpha(z^{\gamma}_1)] = \bigoplus_{\alpha=0}^{k-1} \mathsf{RBF}_S[h_\alpha(z^{\gamma}_2)] = r^{\gamma}; \tag{3}$$

and,

$$\bigoplus_{\alpha=0}^{k-1} \mathsf{RBF}_S[h_\alpha(z^{\gamma}_3)] = s^{\gamma}. \tag{4}$$

Moreover, the construction (1) of $z^{\gamma}_0, z^{\gamma}_1, z^{\gamma}_2$, and z^{γ}_3 follows the input columns of AND gate's truth table. Consequently, Bob obtains $z^{\gamma}_l = w^{\gamma}_i || w^{\gamma}_j$ for inputs $w^{\gamma}_i, w^{\gamma}_j$ of γ-th non-input AND gate $G^{\gamma}_{\mathsf{AND}}(\gamma = 1, \ldots, N)$. Thus, he computes r^{γ} or s^{γ} by calculating $\bigoplus_{\alpha=0}^{k-1} \mathsf{RBF}_S[h_\alpha(z^{\gamma}_l)]$. If $z^{\gamma}_l \in \{z^{\gamma}_0, z^{\gamma}_1, z^{\gamma}_2\}$, Bob will obtain r^{γ} and correspondingly the output of non-input AND gate $G^{\gamma}_{\mathsf{AND}}$ is $c^{\gamma}_0 = h(r^{\gamma})$. If $z^{\gamma}_l = z^{\gamma}_3$, Bob will obtain s^{γ} and correspondingly the output of non-input AND gate $G^{\gamma}_{\mathsf{AND}}$ is $c^{\gamma}_1 = h(s^{\gamma})$. Thus, for inputs $w^{\gamma}_i, w^{\gamma}_j$, we can see that Bob obtains $c^{\gamma}_{\sigma} = c^{\gamma}_{i.j}$ correctly as per the truth table of an AND gate. In particular, if $w^{\gamma}_i \in \{a^{\gamma}_0, a^{\gamma}_1\}, w^{\gamma}_j = b^{\gamma}_0$, and if $w^{\gamma}_i = a^{\gamma}_0, w^{\gamma}_j = b^{\gamma}_1$,

then from (3) it follows that Bob calculates r^γ and gets $c_\sigma^\gamma = h(r^\gamma) = c_0^\gamma = c_{\{i.j\}}^\gamma$ correctly. Similarly, if $w_i^\gamma = a_1^\gamma, w_j^\gamma = b_1^\gamma$, from (4) it follows that Bob calculates s^γ and obtains $c_\sigma^\gamma = h(s^\gamma) = c_1^\gamma = c_{\{1.1\}}^\gamma$. Therefore, Bob correctly obtains bit strings $c_\sigma^\gamma(\gamma = 1, \ldots, N)$ 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 $G_{\mathsf{NOT}}^\gamma(\gamma = 1, \ldots, M)$, Bob has bit strings w_i^γ corresponding to the input wire. To compute $G_{\mathsf{NOT}}^\gamma(\gamma = 1, \ldots, M)$, Alice sends the set $\{\delta^\gamma\}_{\gamma=1}^M$ to Bob. She evaluates $\Delta^\gamma = h(\delta^\gamma)$ for $\gamma = 1, \ldots, M$. According to our construction 3.1, the output of non-input NOT gates is set to be $c_0^\gamma = a_1^\gamma \oplus \Delta^\gamma$ associated with the bit value 0 or $c_1^\gamma = a_0^\gamma \oplus \Delta^\gamma$ associated with the bit value 1. a_0^γ is a bit string corresponding to 0, and a_1^γ be the bit strings corresponding to 1 associated with input wire of each G_{NOT}^γ. Consequently for $G_{\mathsf{NOT}}^\gamma(\gamma = 1, \ldots, M)$, Bob computes $c_0^\gamma = w_i^\gamma \oplus h(\delta^\gamma) = w_i^\gamma \oplus \Delta^\gamma$, when $w_i^\gamma = a_1^\gamma$ and $c_1^\gamma = w_i^\gamma \oplus h(\delta^\gamma) = w_i^\gamma \oplus \Delta^\gamma$, when $w_i^\gamma = a_0^\gamma$. Thus, for input w_i^γ, we can see from the above data that Bob obtains $c_\sigma^\gamma = c_{\neg i}^\gamma$ correctly as per the truth table of NOT gate. In particular, if $w_i^\gamma = a_0^\gamma$, Bob correctly obtains $c_\sigma^\gamma = c_1^\gamma = c_{\{\neg 0\}}^\gamma$. If $w_i^\gamma = a_1^\gamma$, Bob correctly obtains $c_\sigma^\gamma = c_0^\gamma = c_{\{\neg 1\}}^\gamma$. Therefore, Bob correctly obtains the bit string $c_\sigma^\gamma(\gamma = 1, \ldots, M)$ 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 the associated bit value of that bit string correctly.

Table 1. List of Symbols

Symbol	Meaning
x	Private input of Alice
y	Private input of Bob
v_1	Length of x
v_2	Length of y
u	Length of output of function f
λ	Length of arbitrary bit strings corresponding to input values 0 and 1, chosen by Alice
η	Length of bit strings at each entries of RBF_S
k	Number of hash functions used to construct RBF_S
m	Optimal length of RBF_S
q	Index of Alice's input
p	Index of Bob's input
v	Number of input gates in the circuit C
N	Number of non-input AND gates in the circuit C
M	Number of non-input NOT gates in the circuit C
$\in_R$	Randomly chosen
w_r	Corresponding bit string of output wire
c_σ	Corresponding value of final output wire, obtained by Bob
$\stackrel{c}{\equiv}$	Indistinguishable symbol
$\|\|$	Concatenation
$\stackrel{R}{\leftarrow}$	Randomly selected

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.

- **Bob's privacy:** Alice should not be able to obtain any information about Bob's private inputs.
- **Alice's privacy:** Bob learns only the information about Alice's input that he can already compute from the output and his own private input.

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

- The underlying OT protocol utilized to instantiate BooleanEval is secure.
- The hash functions used in our protocol are cryptographically secure.
- Alice and Bob behave in a semi-honest manner [8]. That is, Alice and Bob execute the protocol honestly, but they may try to obtain private information from the messages they receive from other parties.

We will provide a simulation-based security proof of BooleanEval in real-ideal paradigm. We follow the approach of [14]. Intuitively, the protocol ϕ that computes a function $f(x, y)$, 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 View_A^ϕ and the view of Bob to protocol ϕ as View_B^ϕ. That is, View_A^ϕ represents Alice's private inputs, random choices, and the messages received by Alice during the execution of the protocol. Similarly, View_B^ϕ represents Bob's private inputs, random choices, and the messages received by Bob during the execution of the protocol. Consider two simulators $\mathsf{Sim}_1, \mathsf{Sim}_2$ that provide output for the view of Alice and Bob in the ideal world. In particular, Sim_1 and Sim_2 respectively denote the simulated views of Alice and Bob, which contain inputs of the corresponding party, simulated random choices, and simulated protocol messages received by the corresponding party. We are going to show that $\{\mathsf{Sim}_1(x, f(x, y))\} \stackrel{c}{\equiv} \mathsf{View}_A^\phi$ and $\{\mathsf{Sim}_2(y, f(x, y))\} \stackrel{c}{\equiv} \mathsf{View}_B^\phi$. That is, the views of Alice and Bob in the ideal world are indistinguishable from their views in the real world.

Theorem 2. *Alice should be unable to obtain any information about Bob's private inputs.*

Proof. Let us assume that Alice is corrupted. Simulator Sim_1 is constructed in such a way that given input $(x, f(x, y))$, Sim_1 can simulate the view of Alice in the ideal world. Alice has private inputs x_q 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 $\mathsf{Sim}_1^{\mathsf{OT}}$ denote the simulator for the secure 1-out-of-2 OT protocol for input gates. Alice's input for the OT protocol is $(a_0, a_1) \in_R \{0,1\}^\lambda$ (for input NOT gate) or $(a_0, \bar{x}_q a_0 \oplus x_q a_1)$ (for input AND gate). We assume that the OT protocol used in this SFE scheme is secure, i.e, Alice can't obtain the choice bit of Bob. Thus, when the simulator Sim_1 computes the OT protocol with Alice's inputs, it is impossible to gather extra information about the choice bit, i.e., about the private inputs y_p of Bob. Therefore, it is easy to conclude that the output of Sim_1 in ideal world is indistinguishable from the view of Alice in real world, i.e., $\{\mathsf{Sim}_1(x, f(x, y))\} \stackrel{c}{\equiv} \mathsf{View}_A^\phi$. Thus, the protocol preserves the privacy of Bob's private inputs.

Theorem 3. *Bob learns only the information about Alice's input that he can already compute from the output and his own private input.*

Proof. We construct a simulator Sim_2 such that given input $(y, f(x,y))$ to Sim_2, it can simulate the view of Bob in the ideal world. Bob's view of this protocol consists of the OT protocol, RBF_S, and the set $\{\delta^\gamma\}_{\gamma=1}^M$. Let us consider simulator $\mathsf{Sim}_2^{\mathsf{OT}}$ that simulates the OT protocol for Bob. The simulator $\mathsf{Sim}_2^{\mathsf{OT}}$ takes input y_p and outputs a_0 or $a_1 \in_R \{0,1\}^\lambda$ (for input NOT and input AND gates). Note that Alice's input for OT is arbitrary bit strings a_0 and a_1 associated with x_q's. However, $\mathsf{Sim}_2^{\mathsf{OT}}$ does not know Alice's inputs x_q due to the security property of the OT protocol. Thus, Sim_2 has no additional information about Alice's private inputs other than Bob's chosen message. Now, Sim_2 has outputs obtained from $\mathsf{Sim}_2^{\mathsf{OT}}$. The simulator Sim_2 must now construct the RBF and the set of deltas. For the non-input NOT gates, since Alice sends a set of random strings $\{\delta^\gamma\}_{\gamma=1}^M$ to Bob, the simulator can simply generate a new set of random strings $\{\delta'^\gamma\}_{\gamma=1}^M$ of length η and send them to the simulated Bob. This is computationally indistinguishable from the real-world view. The simulator for the RBF is more complex. The simulator knows the correct output of the circuit and can therefore determine the correct wire labels for Bob. For non-input AND gates, the simulator computes the correct output wire label $h(\rho^\gamma)$ for the gate G_{AND}^γ. The simulator then generates a randomized Bloom filter RBF'_S by inserting random shares that correctly produce the value ρ^γ that Bob would expect to see based on his inputs. For any other input combination that Bob could test (which is not his correct input), the simulator ensures that the XOR sum from the RBF is a random value that does not correspond to a valid output for that gate. This is achieved by generating the RBF entries from random bits, with the constraint that the XOR sum for the correct input path equals ρ^γ. Since Bob cannot distinguish between a truly random RBF and a simulated one, the simulation is successful. Therefore, Sim_2 uses random strings to simulate the randomized Bloom filter as is done in the real world. So, the view of Bob in the ideal world that is simulated by Sim_2 is computationally indistinguishable from the real world execution. Moreover, since the set $\{\delta^\gamma\}_{\gamma=1}^M$ consists of only random strings of length η, the simulated view of Bob to the set $\{\delta^\gamma\}_{\gamma=1}^M$ is computationally indistinguishable from the real view. Thus, Sim_2 is indistinguishable from the real view of Bob i.e., $\{\mathsf{Sim}_2(y, f(x,y))\} \stackrel{c}{\equiv} \mathsf{View}_B^\phi$. Therefore, Bob can not gain any information about Alice's private inputs (x_q). Thus, Bob learns only the information about Alice's input that he can already compute from the output and his own private input. Thus, the protocol preserves the privacy of Alice's private inputs.

5 Efficiency and Comparison

We now present the efficiency analysis of $\mathsf{BooleanEval}$. The proposed scheme utilizes 1-out-of-2 OT protocol as a building block to compute input gates. In addition, $\mathsf{BooleanEval}$ employs randomized Bloom filter (RBF) and hash evaluations to compute the final output of a circuit C. Let us assume that to construct the circuit C, N number of non-input AND gates $G_{\mathsf{AND}}^\gamma(\gamma = 1, 2, \ldots N)$ and M number of non-input NOT gates $G_{\mathsf{NOT}}^\gamma(\gamma = 1, 2, \ldots M)$ are required. The complexity of $\mathsf{BooleanEval}$ depends on the computation and communication complexity of the underlying OT used to instantiate $\mathsf{BooleanEval}$. Let us assume that the underlying OT's communication and computation costs are ω_1 and ω_2, respectively. Let the number of input gates in the circuit C be v.

Communication Complexity. A total of v_2 OT executions are required to evaluate a circuit C using $\mathsf{BooleanEval}$. Thus, the total communication cost that occurred during OT execution is $v_2\omega_1$. There are $v - v_2$ input NOT gates corresponding to Alice's input. λ bits are required to send the random bit strings a_i to Bob for each of such input NOT gates. Thus, a total of $(v - v_2)\lambda$ bits are

required for this step. To compute non-input AND gates, an array RBF_S of optimal length m is transmitted to Bob. RBF_S has m entries, each of length η. Therefore, the communication cost to send RBF_S to Bob is $m\eta$. Moreover, to send $\{\delta^\gamma\}_{\gamma=1}^{M}$ to Bob for computing non-input NOT gate, we require $M\eta$ bits since each δ^γ is a string of length η. Note that the communication complexity for non-input gates is independent of λ, i.e., the length of arbitrary strings associated with input values 0 and 1. In total, the communication complexity for BooleanEval is $v_2\omega_1 + (v - v_2)\lambda + (m + M)\eta$.

Table 2. Comparison summary of SFE protocols

Protocol	Commitment schemes	Call for hash/ AND gate		XOR operations/ AND gate		Communication cost/ AND gate		Assumption
		Input	Non-input	Input	Non-input	Input	Non-input	
Malkhi et al. [16]	Applied	12	12	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda)$	$4(\lambda + 1)+\omega_1$	$4(\lambda + 1)+\omega_1$	PRF
Kolesnikov et al. [14]	Not applied	5	5	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda)$	$4(\lambda + 1)+\omega_1$	$4(\lambda + 1)+\omega_1$	CCR
Zahur et al. [27]	Not applied	6	6	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda)$	$(2\lambda)+\omega_1$	$(2\lambda)+\omega_1$	CCR
Rosulek et al. [21]	Not applied	9	9	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda)$	$(1.5\lambda + 5)+\omega_1$	$(1.5\lambda + 5)+\omega_1$	CCR
Pinkas et al. [18] (method-I)	Applied	7	7	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda)$	$3(\lambda + 1)+\omega_1$	$3(\lambda + 1)+\omega_1$	CCR
Pinkas et al. [18] (method-II)	Applied	10	10	$\mathcal{O}(\lambda)$ (addition in $\mathbb{F}_{2^\lambda}$)	$\mathcal{O}(\lambda)$ (addition in $\mathbb{F}_{2^\lambda}$)	$(2\lambda + 4)+\omega_1$	$(2\lambda + 4)+\omega_1$	PRF
BooleanEval	Not applied	0	$5k + 3$	λ	$7(k - 1)\eta$ (independent of λ)	ω_1	$m\eta$ (independent of λ)	PRF

N = Number of non-input AND gates in circuit C, M = Number of non-input NOT gates in C, λ = Length of arbitrary bit strings (at least 256 bits) corresponding to input values 0 and 1, η= Length of bit strings (sufficient to choose 2 or 3 bits) at each entries of RBF_S, k= Number of hash functions used to construct RBF_S (sufficient to take $k = 20$), m= Optimal length ($1.44kn = 5.76kN$) of RBF_S,ω_1= communication cost of the underlying OT protocol, CCR = Circular correlation robust hash function [10], PRF= Pseudorandom hash function.

Computation Complexity. Alice performs λ XOR operations for her input message $(a_0, \bar{x}_q a_0 \oplus x_q a_1)$ to the underlying OT protocol. Since a total of v_2 OT executions are required, the total cost incurred during OT execution is $v_2\omega_2$. The computation of each non-input AND gate requires $4k$ hash evaluations to be done by Alice during insertion and k hash evaluations to be done by

Table 3. Comparison summary of communication cost per non-input AND gate for a circuit C corresponding to SFE protocols

Protocol	Malkhi et al. [16]	Kolesnikov et al. [14]	Zahur et al. [27]	Rosulek et al. [21]	Pinkas et al. [18] (method-I)	Pinkas et al. [18] (method-II)	BooleanEval
Commun. cost/non-input AND gate (in bits)	$(4(\lambda + 1) + \omega_1) = 1028 + \omega_1$	$(4(\lambda + 1) + \omega_1) = 1028 + \omega_1$	$(2\lambda + \omega_1) = 512 + \omega_1$	$((1.5\lambda + 5) + \omega_1) = 389 + \omega_1$	$(3(\lambda + 1) + \omega_1) = 771 + \omega_1$	$((2\lambda + 4) + \omega_1) = 516 + \omega_1$	$m\eta$= 231 for $\eta = 2$ 346 for $\eta = 3$

ω_1= communication cost of the underlying OT protocol, η= Length of bit strings (sufficient to choose 2 or 3 bits) at each entries of RBF_S, $\lambda = 256$ bits.

Bob for retrieval. Moreover, to set the output of each non-input AND gate, Alice needs 2 hash evaluations, and to obtain the output of each non-input AND gate, Bob needs 1 hash evaluation. Thus, the number of hash evaluations for each non-input AND gate is $4k + 2$ for Alice and $k + 1$ for Bob. In addition, each share $t^{\gamma}_{l\alpha}$ and $\widetilde{t}^{\gamma}_{3\alpha}$ of r^{γ} and s^{γ} respectively, is a bit string of length η for $\alpha = 0, 1, \ldots, k-1; l = 0, 1, 2$. Thus, Alice needs to perform $(k-1)\eta$ XORs to insert the final shares and Bob needs to perform $(k-1)\eta$ XORs to obtain $\rho^{\gamma} \in \{r^{\gamma}, s^{\gamma}\}$. Therefore, Alice performs $6(k-1)\eta$ XOR operations to compute the shares for insertion, while Bob performs $(k-1)\eta$ XOR operations to retrieve the value. In total, the number of XORs for all non-input AND gates is $6(k-1)\eta N$ for Alice and $(k-1)\eta N$ for Bob and the number of hash evaluations is $(4k+2)N$ for Alice and $(k+1)N$ for Bob. The computation of M non-input NOT gates of the circuit C requires M hash evaluations for Alice and M for Bob. Alice needs to perform $2M\lambda$ XOR operations, while Bob needs $M\lambda$ XORs.

We present a detailed comparative analysis of BooleanEval with the existing state-of-the-art protocols. The summary of our comparative analysis is provided in Table 2. The related protocols [16,14,27,21,18] described the garbling procedures of AND and XOR gates in a circuit C. In our approach, utilizing the fact that AND and NOT together are known as universal logic gates, we design the circuit C using only AND and NOT gates. Moreover, we analyze the communication and computation cost for each of the AND and NOT gates. Thus, to compare the efficiency of our scheme with the related state-of-the-art, we compare the communication and computation cost only for AND gates.

As evident by the data in Table 2, BooleanEval is efficient compared to [16,14,27,21,18]. Unlike BooleanEval, [16] requires costly operations, such as commitment schemes. Moreover, we consider the hash functions as pseudorandom functions (PRFs), whereas the protocols proposed in [14,27,21,18] (method-I) assume circular correlation robust (CCR) hash functions [10]. PRF-based hash functions are efficient in practice and hardware accelerated (i.e., using AES), but CCR-based hash functions have fewer known standard implementations. More efficiently, BooleanEval has the communication cost for all non-input AND gates is $m\eta$ that is independent of λ, the length of arbitrary strings associated with input values 0 and 1 in the circuit C. Note that the optimal value of m is computed by $\frac{nk}{\ln 2} \approx 1.44nk$, where n is the cardinality of the set S and k is the number of independent hash

functions, used to compute $\mathsf{RBF_S}$. In our protocol, $n = 4N$, where N is the number of non-input AND gates in the circuit C. Therefore, to calculate the communication cost for each non-input AND gate, $N = 1$ i.e., $n = 4$. The maximum false positive probability of the Bloom filter BF_S corresponding to RBF_S is upper bounded by $\frac{1}{2^k}$. Thus, for a negligible false positive rate, we can choose $k = 20$ for practical implementation [9] of our protocol BooleanEval. Since η is the length of each element of RBF_S, we can choose $\eta = 2$ or 3. Therefore, to calculate the communication cost for each non-input AND gate, the values of the parameters are set as $n = 4, k = 20, \eta = 2$ or 3. Consequently, for each non-input AND gate, the communication cost is $m\eta = 1.44 \times 4 \times 20 \times 2 \approx 231$ if $\eta = 2$ and $m\eta = 1.44 \times 3 \times 20 \times 3 \approx 346$ if $\eta = 3$. However, prior works require communication complexity linear in λ per AND gate. In particular, [16,14] requires $4(\lambda + 1)$-bits, [27] needs 2λ-bits, [21] needs $(1.5\lambda + 5)$-bits, [18] (method-I) requires $3(\lambda + 1)$-bits, and [18] (method-II) requires $(2\lambda + 4)$-bits for each AND gate. Moreover, the XOR operations per AND gate in [16,14,27,21,18] are linear in λ. Since λ is the length of arbitrary bit strings corresponding to the input values 0 and 1, for a typical security level, we can choose $\lambda = 256$ for practical implementation. Therefore, the communication cost of each non-input AND gate among the existing state-of-the-art for the most efficient protocol [21] is $1.5\lambda + 5 = 389$. The comparison analysis of communication complexity for each non-input AND gate corresponding to SFE protocols is shown in Table 3. Thus, the proposed protocol BooleanEval performs better over the existing protocols from the communication point of view. In contrast to the existing schemes, the number of XOR operations for each non-input AND gate in our protocol does not rely on λ as it is $7(k-1)\eta$. However, unlike these prior works, our protocol requires a higher number of hash evaluations that effects the computation cost of BooleanEval. In more detail, we have discussed the difference between the choices of parameters i.e., λ, k and η and their scalability in next section 5.1.

We evaluated the proposed BooleanEval algorithm in Python on the Google Colab platform using Jupyter notebooks. The implementation result of proposed BooleanEval is given in Table 4

Table 4. Implementation result of proposed BooleanEval

Parameters	Input AND gate		Non-input AND gate	
	Runtime	Memory Consumption	Runtime	Memory Consumption
$\lambda = 256, k = 20, \eta = 3, m = 231$	0.00084243 seconds	1.960938 KB	0.00141299 seconds	70.587891 KB

λ = Length of arbitrary bit strings corresponding to input values 0 and 1, k = Number of hash functions used to construct RBF_S, m = Array length of RBF_S, η = Length of bit strings at each entry of RBF_S, KB = Kilobyte.

5.1 Parameter Selection and Scalability

The choice of security parameter λ and parameter k, η is crucial for the overall security and efficiency of the protocol. Note that λ is the length of an arbitrary bit string corresponding to the input values 0 and 1 of the input wires of a circuit C. η is the length of an arbitrary bit string of an element of m-array RBF_S. In addition, k is the number of hash functions used to construct RBF_S. The false positives probability (i.e., $\bar{z} \notin S$ but $\mathsf{BF}_S[h_\alpha(\bar{z})] = 1$) of the Bloom filter BF_S associated with RBF_S is upper bounded by $\frac{1}{2^k}$. Thus, for practical implementation, we can choose to obtain typical security levels, we can set $\lambda = 256$ bits for wire labels and use $k = 20$ as the number of hash functions and $\eta = 2$ or 3 bits for RBF_S entries to ensure a negligible probability of false positives of the associated Bloom filter BF_S. The size m of the RBF_S is determined by the number of non-input AND gates N and the number of hash functions k with $m \approx 1.44kn$, where $n = 4N$. This means that as the number of gates increases, the size of the RBF_S grows linearly. This linear scaling suggests that the protocol is scalable for larger circuits.

6 Conclusions

In this paper, we presented the design and simulation based security analysis of a two-party one-way SFE protocol BooleanEval. We propose a variant of standard Bloom filter, namely randomized Bloom filter (RBF). The design of SBE utilizes a 1-out-of-2 OT and RBF as a cryptographic building block. BooleanEval is efficient and fast, as the only cost-intensive operations employed in BooleanEval are hash evaluation and XOR operations. Moreover, the communication cost for non-input AND gates does not depend on the security parameter $\lambda(\geq 256$ bits), the length of arbitrary bit strings associated with input values 0 and 1 in the circuit C.

Acknowledgments

Vikas Srivastava acknowledges the support received from the ANRF-PMECRG project with Ref. No. ANRF/ECRG/2025/002808/PMS. The preliminary version of the paper is available at (`https://eprint.iacr.org/2024/1787.pdf`). The paper was edited for grammar using "Grammarly" (`https://www.grammarly.com/`).

References

1. 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. Springer (2011)
2. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM **13**(7), 422–426 (1970)
3. 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)
4. Chou, T., Orlandi, C.: The simplest protocol for oblivious transfer. In: International Conference on Cryptology and Information Security in Latin America. pp. 40–58. Springer (2015)

5. 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. Springer (2014)
6. Dong, C., Chen, L., Wen, Z.: When private set intersection meets big data: an efficient and scalable protocol. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. pp. 789–800 (2013)
7. 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 Machine Learning (ICML 2016). Proceedings of Machine Learning Research, vol. 48, pp. 201–210. JMLR.org, New York, NY, USA (2016)
8. Evans, D., Kolesnikov, V., Rosulek, M., et al.: A pragmatic introduction to secure multi-party computation. Foundations and Trends® in Privacy and Security **2**(2-3), 70–246 (2018)
9. Fan, L., Cao, P., Almeida, J., Broder, A.Z.: Summary cache: a scalable wide-area web cache sharing protocol. IEEE/ACM transactions on networking **8**(3), 281–293 (2000)
10. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Annual International Cryptology Conference. pp. 145–161. Springer (2003)
11. 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. Springer (2000)
12. 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)
13. Kolesnikov, V., Sadeghi, A.R., Schneider, T.: A systematic approach to practically efficient general two-party secure function evaluation protocols and their modular design. Journal of Computer Security **21**(2), 283–315 (2013)
14. 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. Springer (2008)
15. Lindell, Y.: Secure multiparty computation for privacy preserving data mining. In: Encyclopedia of data warehousing and mining, pp. 1005–1009. IGI Global Scientific Publishing (2005)
16. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y., et al.: Fairplay-secure two-party computation system. In: USENIX security symposium. vol. 4, p. 9. San Diego, CA, USA (2004)
17. Mano, M.M.: Digital logic and computer design. Pearson Education India, New Delhi (2017)
18. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: Advances in Cryptology–ASIACRYPT 2009: 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings 15. pp. 250–267. Springer (2009)
19. 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)
20. 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)
21. Rosulek, M., Roy, L.: Three halves make a whole? beating the half-gates lower bound for garbled circuits. In: Annual International Cryptology Conference. pp. 94–124. Springer (2021)
22. 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)
23. Schneider, T.: Practical secure function evaluation. In: Informatiktage. pp. 37–40 (2008)

24. 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)
25. Yao, A.C.: Protocols for secure computations. In: 23rd annual symposium on foundations of computer science (sfcs 1982). pp. 160–164. IEEE (1982)
26. Yao, A.C.C.: How to generate and exchange secrets. In: 27th annual symposium on foundations of computer science (Sfcs 1986). pp. 162–167. IEEE (1986)
27. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole: Reducing data transfer in garbled circuits using half gates. In: Advances in Cryptology-EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II 34. pp. 220–250. Springer (2015)

A Boolean Function

A Boolean function [17] 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, and 2.

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

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

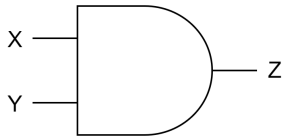
Example 2. $f_{\mathsf{NOT}} : \{0,1\} \longrightarrow \{0,1\}$ is defined by

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

B Universal Logic Gates

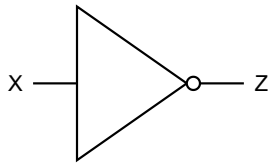
Logic gates are a fundamental tool for constructing a digital circuit. Basic logic gates are $\mathsf{AND}, \mathsf{NOT}$ gates. In general, $\mathsf{AND}, \mathsf{NOT}$ operations are represented as $\cdot, \neg$ respectively. The constructions and truth tables of these gates are given in Example 3 and 4.

Example 3. AND gate construction and its truth table.



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

Example 4. NOT gate construction and its truth table.



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

Note that the truth tables of the logic gates AND, NOT represents the corresponding Boolean function's output (refer Example 1, 2). 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.

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 3.

- *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 3.
- *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 3.

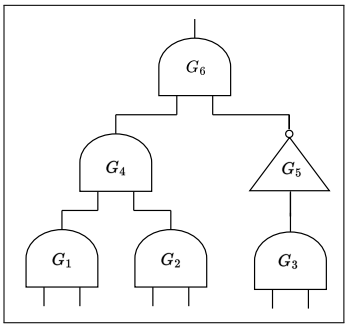


Fig. 3. Example of input and non-input gates of a circuit C

C 1-out-of-2 Oblivious Transfer

1-out-of-2 oblivious transfer (OT) [4] is a two-party privacy-preserving cryptographic protocol. It involves two parties, say Alice and Bob, where Alice has a message pair (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 4. 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.

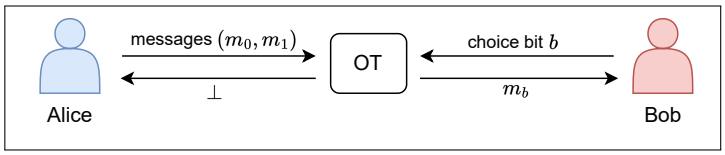


Fig. 4. 1-out-of-2 Oblivious Transfer

D A Toy Example

In this section, we provide a toy example of BooleanEval. Let us consider a Boolean function $f : \{0,1\}^2 \times \{0,1\}^2 \to \{0,1\}$ defined by $f(x,y) = [\{\neg(x_1 \cdot y_1) + x_2\} \cdot (\neg y_2)]$. Here, '$\cdot$' symbolizes the logical conjunction (AND gate), '$\neg$' symbolizes logical negation (NOT gate), and '$+$' symbolizes logical disjunction (OR gate). Using De Morgan's law we have,

$$\begin{aligned} f(x,y) &= [\{\neg(x_1 \cdot y_1) + x_2\} \cdot (\neg y_2)] \\ &= [\neg\{(x_1 \cdot y_1) \cdot (\neg x_2)\} \cdot (\neg y_2)] \quad \text{[by De Morgan's 2nd law]}. \end{aligned}$$

The Boolean circuit C corresponding to $f(x,y)$ is given in Figure 5. Alice has private input $x = (1,0)$ and Bob has private input $y = (1,1)$. Here, $v_1 = 2, v_2 = 2, u = 1$ and $x_1 = 1, x_2 = 0$; $y_1 = 1, y_2 = 1$. RBF_S is randomized Bloom filter parameterized with (n, m, k, H, η). Since there are two non-input AND gates in the Figure 5, $n = 8$. Let $k = 3$, then the optimal length of RBF_S is $m = \frac{nk}{\ln 2} \approx 34$. H be the set of $k = 3$ hash functions defined by $\{h_\alpha : \{0,1\}^* \longrightarrow \{0,1,\ldots,33\} \mid \alpha = 0,1,2\}$. Also assume that $h : \{0,1\}^\eta \longrightarrow \{0,1\}^\lambda$ be a cryptographically secure hash function, where η, λ are security parameters. Let W_i be the output wire of G_i for $i = 1,2,\ldots,6$. We denote w_i be the bit string obtained by Bob corresponding to the output wire W_i of gate G_i for $i = 1,2,\ldots,6$. The computation of the circuit is described below.

Computation of Input Gates (G_1, G_2, G_3):

– Alice has private inputs $x_1 = 1$ and Bob has private inputs $y_1 = 1$ for G_1 (AND gate). Bob has private input $y_2 = 1$ for G_3 (NOT gate), and Alice has private inputs $x_2 = 0$ for G_2 (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 a 1-out-of-2 OT. Since $x_1 = 1$, Alice's input messages to OT are $(a_0^1, \bar{x}_1 a_0^1 \oplus x_1 a_1^1) = (a_0^1, a_1^1)$ and Bob's choice bit is $y_1 = 1$. Thus, Bob receives a_1^1 as an output of OT. Therefore $w_1 = a_1^1$.
– G_2 is an input NOT with Alice's input $x_2 = 0$. Alice chooses $a_0^2, a_1^2 \in_R \{0,1\}^\lambda$ associated with 0 and 1 respectively. Alice sends $a_{\bar{x}_2}^2 = a_1^2$ to Bob. Therefore, $w_2 = a_1^2$.
– G_3 is an input NOT gate in which Bob's input is $y_2 = 1$. Alice first chooses $a_0^3, a_1^3 \in_R \{0,1\}^\lambda$ associated with 0 and 1 respectively. Then Alice and Bob are involved in 1-out-of-2 OT with Alice's input (a_0^3, a_1^3) and Bob's choice bit $\bar{y}_2 = 0$. On completion of OT, Bob receives $w_3 = a_{\bar{y}_2}^3 = a_0^3$.

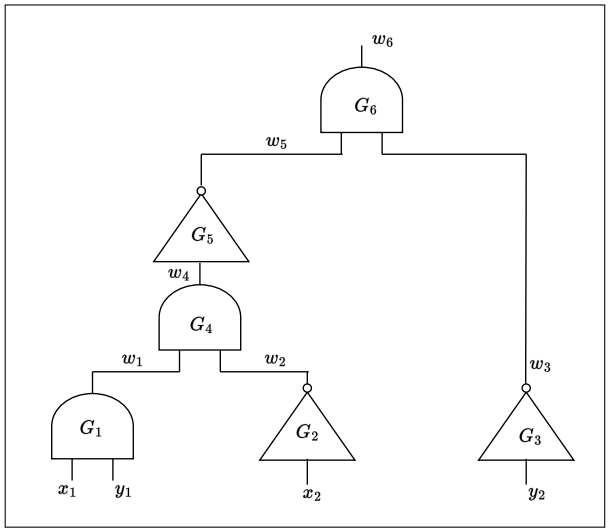


Fig. 5. Flow Diagram of Corresponding Boolean Circuit of Function f

Computation of Non-input Gates (G_4, G_5, G_6)**:** Note that among the non-input gates, there are two non-input AND gates G_4, G_6, and one non-input NOT gate G_5. Moreover, each non-input AND gate (G_4, G_6) have two input wires W_i, W_j with inputs $w_i \in \{a_0^i, a_1^i\}$ and $w_j \in \{a_0^j, a_1^j\}$, where $a_0^i, a_0^j \in \{0,1\}^\lambda$ are associated with 0's and a_1^i, a_1^j are associated with 1's. G_4 has input wires W_1 and W_2. G_6 has input wires W_5 and W_3. The non-input NOT gate (G_5) has input wire W_4 with input $w_4 \in \{a_0^4, a_1^4\}$, where a_0^4 associated with 0 and a_1^4 associated with 1.

– Alice will compute a randomized Bloom filter (namely, RBF_S) for G_4, G_6 as follows:

(i) Computes

$$a_0^1 \,||\, a_0^2 = z_0^4; \quad a_0^1 \,||\, a_1^2 = z_1^4; \quad a_1^1 \,||\, a_0^2 = z_2^4; \quad a_1^1 \,||\, a_1^2 = z_3^4.$$
$$a_0^5 \,||\, a_0^3 = z_0^6; \quad a_0^5 \,||\, a_1^3 = z_1^6; \quad a_1^5 \,||\, a_0^3 = z_2^6; \quad a_1^5 \,||\, a_1^3 = z_3^6.$$

for non-input AND gates G_4, G_6. Writes $S = \{z_0^4, z_1^4, z_2^4, z_3^4, z_0^6, z_1^6, z_2^6, z_3^6\}$.

(ii) Chooses $r^4, s^4, r^6, s^6 \in_R \{0,1\}^\eta$ for non-input AND gates $G_i (i = 4, 6)$, where r^i is associated with each element of $\{z_0^i, z_1^i, z_2^i\}$ and s^i is associated with $\{z_3^i\}$ for $i = 4, 6$ such that $r^4 \neq s^4$ and $r^6 \neq s^6$.

(iii) Sets the output of G_4, G_6 as follows:

$$a_0^4 = h(r^4), a_0^6 = h(r^6) \in \{0,1\}^\lambda, \text{ corresponding to bit 0}$$
$$a_1^4 = h(s^4), a_1^6 = h(s^6) \in \{0,1\}^\lambda, \text{ corresponding to bit 1.}$$

(iv) Constructs an array of optimal length 34, initially filled with a string of zeros $\perp$.

$\mathsf{RBF}_S =$

$\perp$	$\perp$	$\perp$	$\perp$	...	$\perp$	$\perp$
0	1	2	3	...	32	33

(v) Let shares $\{t^4_{0\alpha}\}^2_{\alpha=0}, \{t^4_{1\alpha}\}^2_{\alpha=0}$, and $\{t^4_{2\alpha}\}^2_{\alpha=0} \in_R \{0,1\}^\eta$ are associated with each element of $\{z^4_0, z^4_1, z^4_2\}$ and shares $\{t^4_{3\alpha}\}^2_{\alpha=0} \in_R \{0,1\}^\eta$ are associated with $\{z^4_3\}$. In addition, let the shares $\{t^6_{0\alpha}\}^2_{\alpha=0}, \{t^6_{1\alpha}\}^2_{\alpha=0}$, and $\{t^6_{2\alpha}\}^2_{\alpha=0} \in_R \{0,1\}^\eta$ are associated with each element of $\{z^6_0, z^6_1, z^6_2\}$ and shares $\{t^6_{3\alpha}\}^2_{\alpha=0} \in \{0,1\}^\eta$ are associated with $\{z^6_3\}$ as the following holds:

$$\begin{aligned} r^4 &= t^4_{00} \oplus t^4_{01} \oplus t^4_{02}; & r^6 &= t^6_{00} \oplus t^6_{01} \oplus t^6_{02}; \\ &= t^4_{10} \oplus t^4_{11} \oplus t^4_{12}; & &= t^6_{10} \oplus t^6_{11} \oplus t^6_{12}; \\ &= t^4_{20} \oplus t^4_{21} \oplus t^4_{22}; & &= t^6_{20} \oplus t^6_{21} \oplus t^6_{22}; \\ s^6 &= t^4_{30} \oplus t^4_{31} \oplus t^4_{32}; & s^6 &= t^6_{30} \oplus t^6_{31} \oplus t^6_{32}. \end{aligned}$$

Alice inserts these shares into the array at the corresponding indices. For example, for z^4_0, she inserts t^4_{00} at index $h_0(z^4_0)$, t^4_{01} at $h_1(z^4_0)$, and t^4_{02} at $h_2(z^4_0)$. Let, $h_1(z^4_2) = 7 = h_2(z^4_1)$, $h_1(z^4_3) = 2 = h_0(z^4_1)$, $h_1(z^6_0) = 5 = h_2(z^4_2)$, $h_1(z^6_1) = 0 = h_0(z^4_0)$, $h_2(z^6_2) = 6 = h_0(z^6_1)$, $h_1(z^6_3) = 9 = h_2(z^6_0)$, therefore Alice takes $t^4_{21} = t^4_{12}$, $t^4_{31} = t^4_{10}$, $t^6_{01} = t^4_{22}$, $t^6_{11} = t^4_{00}$, $t^6_{22} = t^6_{10}$, $t^6_{31} = t^6_{02}$. Moreover, let $h_0(z^4_2) = 3, h_0(z^4_3) = 1, h_1(z^4_0) = 32, h_1(z^4_1) = 10, h_2(z^4_0) = 4, h_2(z^4_3) = 8, h_0(z^6_0) = 31, h_0(z^6_2) = 11, h_0(z^6_3) = 17, h_1(z^6_2) = 15, h_2(z^6_1) = 33, h_2(z^6_3) = 30$. Thus, the randomized Bloom filter RBF_S takes the following form:

t^4_{00}	t^4_{30}	t^4_{10}	t^4_{20}	t^4_{02}	t^4_{22}	t^6_{10}	t^4_{12}	t^4_{32}	t^6_{02}	t^4_{11}	t^6_{20}	...	t^6_{11}	$\perp$	t^6_{30}	...	t^6_{32}	t^6_{00}	t^4_{01}	t^6_{12}
0	1	2	3	4	5	6	7	8	9	10	11	...	15	16	17	...	30	31	32	33

(vi) Fills the remaining indices (which are occurring $\perp$) with randomly chosen bit strings $\epsilon_1, \epsilon_2, \ldots, \epsilon_{13}$ of length η. Thus, $\mathsf{RBF}_S =$

t^4_{00}	t^4_{30}	t^4_{10}	t^4_{20}	t^4_{02}	t^4_{22}	t^6_{10}	t^4_{12}	t^4_{32}	t^6_{02}	t^4_{11}	t^6_{20}	ϵ_1	ρ^4_1	ϵ_2	t^6_{11}	ρ^6_1	t^6_{30}	...
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	...

...	ρ^4_2	ϵ_{13}	ρ^6_2	t^6_{32}	t^6_{00}	t^4_{01}	t^6_{12}
...	27	28	29	30	31	32	33

.

– Alice will do the computation of non-input NOT gate G_5 as follows:

(i) Chooses $\delta \in \{0,1\}^\eta$ and computes $\Delta = h(\delta)$.
(ii) Sets the output of G_5 as

$$\begin{aligned} c^4_0 &= a^4_1 \oplus \Delta, \text{ corresponds to the bit 0} \\ c^4_1 &= a^4_0 \oplus \Delta, \text{ corresponds to the bit 1.} \end{aligned}$$

That is, $a^5_0 = c^4_0$ and $a^5_1 = c^4_1$.

Finally, Alice sends RBF_S and δ to Bob. On receiving RBF_S, Bob executes the following:

– Computation for the non-input AND gate G_4.

(i) Since $w_1 = a^1_1$ and $w_2 = a^2_1$, Bob computes $w_1||w_2 = a^1_1||a^2_1 = z^4$. Note that $z^4 = z^4_3$.
(ii) Evaluates $\rho^4 = \overset{2}{\underset{\alpha=0}{\oplus}} \mathsf{RBF}_S[h_\alpha(z^4)]$. Note that $\rho^4 = s^4$.
(iii) Computes the output of G_4 as $h(\rho^4) = h(s^4)$. Note that $h(s^4) = a^4_1$. That is, $w_4 = a^4_1$.

– Computation for the non-input NOT gate G_5.

(i) Computes $\Delta = h(\delta)$ and $w^4 \oplus \Delta = h(\rho^4) \oplus \Delta = h(s^4) \oplus \Delta$. Note that $h(s^4) \oplus \Delta = a_1^4 \oplus \Delta$ corresponds to the bit 0 i.e., a_0^5.
(ii) Sets the output of G^5 as $w^5 = a_1^4 \oplus \Delta$. Note that $w_5 = a_0^5$.

– Computation for the non-input AND gate G_6.

(i) Since $w_5 = a_0^5$ and $w_3 = a_0^3$, Bob computes $w_5||w_3 = a_0^5||a_0^3 = z^6$. Note that $z^6 = z_0^6$.
(ii) Evaluates $\rho^6 = \bigoplus_{\alpha=0}^{2} \mathsf{RBF}_S[h_\alpha(z^6)]$. Note that $\rho^6 = r^6$.
(iii) Computes the output of G_6 as $h(\rho^6) = h(r^6)$. Note that $h(r^6) = a_0^6$. That is, $w_6 = a_0^6$.

Alice sends $(a_0^6, 0)$ and $(a_1^6, 1)$ to Bob. By matching w_6 with a_0^6 and a_1^6, Bob easily understands the output bit of the circuit is 0.