

1 Introduction

We have studied Beaver-triple-based multiplication of secret shares and developed a privacy-preserving inference model for computing $\mathbf{w}^T \mathbf{x} + b$ using two non-colluding, semi-honest servers. However, if one or more servers may behave maliciously, stronger security mechanisms are required. Threshold secret-sharing-based multiplication techniques, such as those used in multiparty computation (MPC), can provide security against a bounded number of malicious parties, but generally require multiple servers and an appropriate threshold configuration.

This raises a natural question: *Can we perform privacy-preserving computation using only one computational server, without requiring multiple servers to hold shares of the private inputs?* Garbled circuits provide an alternative approach. They allow a party to evaluate a function on private inputs by receiving a cryptographically encoded Boolean circuit and suitable input wire labels, with oblivious transfer enabling the evaluator to obtain labels for its own input. Thus, garbled circuits support secure two-party computation without requiring a distributed set of computation servers, although their security guarantees depend on the underlying cryptographic primitives and the assumed adversarial model.

2 Functionality as a Boolean Circuit.

A functionality can be represented as a Boolean circuit if it can be computed using a finite composition of Boolean gates, such as AND, OR, and NOT. Formally, let $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$ be a function. A Boolean circuit C_f computes f if $C_f(x) = f(x)$ for every $x \in \{0, 1\}^n$. For example, equality testing, integer comparison, and arithmetic operations on binary inputs can be implemented using Boolean circuits.

A function is **computable** if an algorithm exists that produces the correct output and terminates for every valid input. However, not every mathematically defined function is computable. For example, the *halting problem*, which asks whether a given program terminates on a given input, is undecidable: no algorithm can answer this question correctly for all programs and inputs. Consequently, no Boolean circuit family can compute the halting function for arbitrary input lengths. A fixed finite circuit computes a function on inputs of a fixed length, whereas computations over arbitrarily long inputs are represented by families of circuits $\{C_n\}_{n \geq 1}$.

For secure computation, *efficient circuit representation* is particularly important. Every polynomial-time computable function admits a polynomial-size circuit family, satisfying $C_n(x) = f_n(x)$ and $|C_n| = \text{poly}(n)$, where $|C_n|$ denotes the number of gates. Although some computable functions may require superpolynomial circuit size, making their evaluation expensive, polynomial-size circuits provide a natural setting for practical garbled-circuit protocols. Thus, the feasibility of garbled circuits depends on whether the desired functionality admits an efficient Boolean circuit representation.

Example: Converting a Function into a Boolean Circuit. Consider the function $f : \{0, 1\}^2 \rightarrow \{0, 1\}$ defined by $f(x_1, x_2) = x_1 \wedge x_2$. This function can be represented by a single AND gate, whose inputs are x_1, x_2 and whose output is $z = x_1 \wedge x_2$. As a more practical example, consider equality testing between two n -bit strings $x = (x_1, \dots, x_n)$ and $y = (y_1, \dots, y_n)$. The equality function $f_{\text{eq}} : \{0, 1\}^{2n} \rightarrow \{0, 1\}$ is defined as

$$f_{\text{eq}}(x, y) = \begin{cases} 1, & \text{if } x = y, \\ 0, & \text{otherwise.} \end{cases}$$

For each $i \in \{1, \dots, n\}$, compute $e_i = \neg(x_i \oplus y_i)$, where $\oplus$ denotes XOR. The output is then

$$z = \bigwedge_{i=1}^n e_i = \bigwedge_{i=1}^n \neg(x_i \oplus y_i).$$

Thus, the equality function can be implemented using n XOR gates, n NOT gates, and $n - 1$ AND gates. The resulting circuit has size $O(n)$. In a garbled-circuit protocol, Alice and Bob can evaluate this circuit on their private strings without directly revealing the strings to each other, subject to the security guarantees of the protocol.

3 Garbled Circuits

Garbled circuits (GC), introduced by Andrew C. Yao [1], enable two parties to compute a function over their private inputs without revealing those inputs to each other, except for information implied by the authorized output. The central idea is to replace each Boolean wire value with a cryptographic label and encrypt the truth table of each gate. The evaluator computes using these labels rather than directly manipulating the underlying Boolean values.

3.1 System and Adversarial Model

Let Alice and Bob hold private inputs $x \in \{0, 1\}^n$ and $y \in \{0, 1\}^m$, respectively. They wish to compute a publicly known function

$$f : \{0, 1\}^n \times \{0, 1\}^m \rightarrow \{0, 1\}^\ell.$$

Let C be a Boolean circuit implementing f . Alice acts as the *garbler*, while Bob acts as the *evaluator*. Alice constructs a garbled version $\tilde{C}$ of C and sends it to Bob, who evaluates it using the labels corresponding to both parties' inputs.

We consider the semi-honest adversarial model, in which both parties follow the protocol but may attempt to infer additional information from their views. The protocol aims to provide correctness and input privacy, relying on a secure garbling scheme and a secure 1-out-of-2 oblivious transfer (OT) protocol.

3.2 Protocol Description

3.2.1 Step 1: Garbling a Single Boolean Gate

We first explain the construction and evaluation of a single AND gate. The same procedure applies to other Boolean gates by replacing the gate's truth function.

Boolean truth table. Consider a gate with input wires u, v and output wire w , where

$$w = u \wedge v.$$

Its truth table is

u	v	$w = u \wedge v$
0	0	0
0	1	0
1	0	0
1	1	1

Ordinarily, evaluating this table requires knowing the input bits. In a garbled circuit, Alice encrypts the table so that Bob can evaluate the gate using cryptographic labels.

Generating wire labels. For every wire q , Alice independently generates two random κ -bit labels:

$$K_q^0, K_q^1 \in \{0, 1\}^\kappa.$$

The label K_q^0 represents Boolean value 0, and K_q^1 represents Boolean value 1. Bob receives only the label corresponding to the actual value on each wire.

To enable efficient table lookup, Alice also associates a permutation bit $p_q^b \in \{0, 1\}$ with each label K_q^b . The two labels on the same wire receive opposite permutation bits:

$$p_q^1 = 1 - p_q^0.$$

The permutation bits are chosen independently of the semantic meaning of the Boolean values. Thus, knowing a permutation bit does not reveal whether its label represents 0 or 1.

Encrypting the truth table. For the AND gate, Alice generates labels

$$K_u^0, K_u^1, \quad K_v^0, K_v^1, \quad K_w^0, K_w^1.$$

For every possible input pair $(a, b) \in \{0, 1\}^2$, she creates an encrypted entry

$$E_{a,b} = \text{Enc}_{K_u^a, K_v^b} \left(K_w^{a \wedge b} \right).$$

Here, Enc denotes the encryption operation of a secure garbling scheme, using both input labels to protect the output label. The gate identifier and other required context may also be incorporated into the encryption to prevent cross-gate misuse.

Alice does not send these entries in their semantic truth-table order. Instead, she places each entry at the position determined by the permutation bits of its input labels:

$$\tilde{T}_g \left[p_u^a, p_v^b \right] = E_{a,b}.$$

Consequently, the four entries occupy the four positions $(0, 0)$, $(0, 1)$, $(1, 0)$, and $(1, 1)$, but their positions do not directly indicate the underlying Boolean input values.

The resulting garbled table can be represented as

Table position	Encrypted entry
$(0, 0)$	$\tilde{T}_g[0, 0]$
$(0, 1)$	$\tilde{T}_g[0, 1]$
$(1, 0)$	$\tilde{T}_g[1, 0]$
$(1, 1)$	$\tilde{T}_g[1, 1]$

The positions are determined by the permutation bits, not by the logical input values. Therefore, the table does not expose the original truth-table ordering.

3.2.2 Step 2: Evaluating a Single Garbled Gate

Suppose the actual input values are $u = a$ and $v = b$. Bob possesses the corresponding active labels K_u^a and K_v^b , but he does not need to know a or b .

Selecting the correct entry. Bob reads the permutation bits attached to his active labels:

$$i = p_u^a, \quad j = p_v^b.$$

He then selects the entry at position (i, j) :

$$E = \tilde{T}_g[i, j].$$

Because Alice placed each encrypted entry according to the permutation bits of its associated input labels, this position contains exactly the entry encrypted for Bob's active labels.

Thus, Bob does not have to identify the logical input pair or try all four entries. He directly locates the correct entry using the two permutation bits.

Decrypting the entry. Bob uses the active input labels to decrypt the selected entry:

$$K_w^{a \wedge b} = \text{Dec}_{K_u^a, K_v^b}(E).$$

The output is the active label for the gate's output wire. Bob does not learn the underlying output bit merely from obtaining this label.

For example, if $a = 1$ and $b = 0$, Bob holds K_u^1 and K_v^0 . He selects the entry at

$$(p_u^1, p_v^0)$$

and obtains K_w^0 after decryption, because $1 \wedge 0 = 0$. The permutation bits tell Bob which entry to select, but do not directly tell him that the logical input pair is $(1, 0)$.

This selection procedure is the key distinction between an ordinary truth table and a point-and-permute

garbled table.

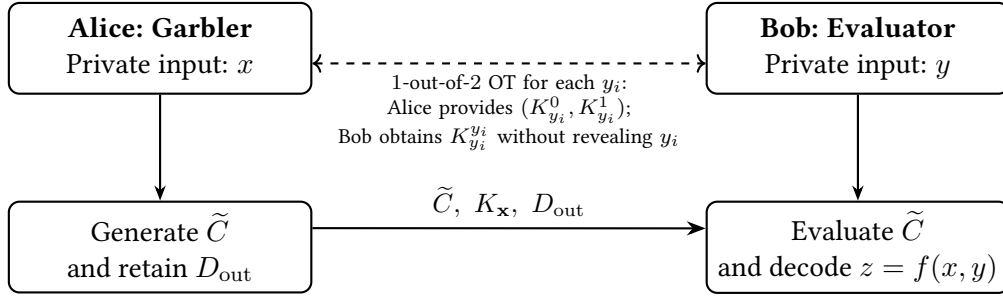


Figure 1: Communication model for two-party garbled-circuit evaluation.

3.2.3 Step 3: Garbling Multiple Gates

Let the Boolean circuit C contain gates $g_1, g_2, \dots, g_t$, arranged in topological order. Alice applies the single-gate construction independently to every gate.

For a gate g_i with input wires u_i, v_i and output wire w_i , she generates a garbled table $\tilde{T}_{g_i}$ satisfying

$$\text{Eval}\left(\tilde{T}_{g_i}, K_{u_i}^a, K_{v_i}^b\right) = K_{w_i}^{g_i(a,b)}.$$

Here, Eval denotes the garbled-gate evaluation procedure: it reads the permutation bits of the active input labels, selects the corresponding table entry, and decrypts it to obtain the active output label.

Alice sends Bob the collection of garbled tables

$$\tilde{C} = \left(\tilde{T}_{g_1}, \tilde{T}_{g_2}, \dots, \tilde{T}_{g_t}\right).$$

Each table is constructed using the labels of its own input and output wires. If two gates share a wire, they use the same pair of labels for that wire. This consistency allows the output label of one gate to serve as an input label to another gate.

3.2.4 Step 4: Evaluating Multiple Layers

Let $C = (W, G)$ be a Boolean circuit, where W denotes the set of wires and $G = \{g_1, \dots, g_t\}$ is the set of gates arranged in topological order. For each gate g_i , let u_i, v_i denote its input wires and w_i its output wire. The corresponding garbled gate is denoted by $\tilde{T}_{g_i}$.

Suppose gate g_i computes

$$w_i = g_i(u_i, v_i).$$

Given the active input labels $K_{u_i}^a$ and $K_{v_i}^b$, Bob evaluates the garbled gate to obtain the active output label:

$$K_{w_i}^{g_i(a,b)} = \text{Eval}\left(\tilde{T}_{g_i}, K_{u_i}^a, K_{v_i}^b\right).$$

Here, Eval selects the appropriate garbled-table entry using the permutation bits of the active input labels and decrypts it to recover the output label.

If w_i is an input wire of a subsequent gate g_j , the label $K_{w_i}^{g_i(a,b)}$ is directly used as the corresponding active input label for evaluating $\tilde{T}_{g_j}$. No additional conversion is required between consecutive gates because all gates sharing a wire use the same pair of wire labels.

Bob evaluates the gates in topological order, beginning with the active labels corresponding to the parties' input bits. At each gate, the active input labels determine the active output label. Consequently, the evaluation proceeds inductively through the circuit until active labels are obtained for all output wires.

Formally, if b_w denotes the Boolean value carried by wire w during the evaluation of C , then Bob computes

its corresponding active label K_w^{bw} for every wire $w \in W$. In particular, for each output wire o_j , he obtains

$$K_{o_j}^{z_j}, \quad z = f(x, y).$$

Thus, Bob evaluates the complete garbled circuit $\tilde{C}$ without explicitly recovering the intermediate Boolean values. The final output labels are decoded using the authorized output decoding information to obtain $z = f(x, y)$.

3.2.5 Step 5: Input Delivery and Output Decoding

Delivering input labels. For Alice's input $x = (x_1, \dots, x_n)$, Alice sends Bob the active labels

$$K_X = (K_{x_1}^{x_1}, \dots, K_{x_n}^{x_n}).$$

For Bob's input $y = (y_1, \dots, y_m)$, Alice and Bob execute a 1-out-of-2 OT protocol for each input bit y_i . Alice acts as the OT sender with messages $(K_{y_i}^0, K_{y_i}^1)$, and Bob acts as the receiver with choice bit y_i . Bob obtains $K_{y_i}^{y_i}$ without revealing his choice to Alice.

Thus, Bob obtains the active labels for all input wires without receiving both labels for any input wire.

Output decoding. After evaluating the circuit, Bob obtains the active label $K_{o_j}^{z_j}$ for each output wire o_j , where $z = f(x, y)$.

For each output wire o_j , Alice can provide the decoding bit

$$d_j = p_{o_j}^0.$$

Since the permutation bits of the two labels on a wire are opposite, Bob recovers the output bit as

$$z_j = p_{o_j}^{z_j} \oplus d_j.$$

Indeed, $p_{o_j}^{z_j} = d_j \oplus z_j$. Therefore, Bob can decode the output without receiving both output labels or learning the labels corresponding to the opposite output values.

The decoding information should be restricted to the authorized outputs. Alice does not provide the semantic mapping for every intermediate wire.

3.3 Complete Protocol Summary

The garbled-circuit protocol proceeds as follows:

1. **Circuit construction:** Alice represents the target function $f(x, y)$ as a Boolean circuit C .
2. **Garbled circuit generation:** Alice generates two cryptographic labels for each wire and constructs a point-and-permute garbled table for every gate in C , producing the garbled circuit $\tilde{C}$.
3. **Garbled circuit transmission:** Alice sends $\tilde{C}$ and the required garbling information to Bob.
4. **Input-label delivery:** Alice sends Bob the active labels corresponding to her input x . Bob obtains the active labels corresponding to his input y through 1-out-of-2 oblivious transfer.
5. **Circuit evaluation:** Bob evaluates the gates of $\tilde{C}$ in topological order. At each gate, he uses the permutation bits of the active input labels to select the appropriate table entry and decrypts it to obtain the active output label.
6. **Output decoding:** Bob applies the authorized output decoding information to the final output labels and recovers $z = f(x, y)$.

3.4 Correctness and Security

Correctness. For every gate g with input wires u, v and output wire w , the garbling construction ensures that

$$\text{Eval}(\tilde{T}_g, K_u^a, K_v^b) = K_w^{g(a,b)}$$

for every $(a, b) \in \{0, 1\}^2$. Since the output label of each gate is the correct active label for its output wire, induction over the gates in topological order establishes that Bob obtains the correct active label for every wire. Output decoding consequently yields

$$\text{Decode}(K_{\text{out}}) = f(x, y).$$

Security Intuition. In the semi-honest model, both parties follow the protocol but may attempt to infer additional information from their views.

- **Privacy of Alice's input:** Bob receives only the active labels for Alice's input wires. The garbled circuit conceals the intermediate values and inactive labels.
- **Privacy of Bob's input:** Oblivious transfer ensures that Alice does not learn Bob's OT choice bits, while Bob obtains only the selected input labels.
- **Output leakage:** The output itself may reveal information about the inputs. Garbled circuits do not eliminate information logically implied by the authorized output.

These guarantees rely on the security of the garbling scheme and OT. The basic protocol described here targets semi-honest adversaries. Malicious security requires additional mechanisms to ensure that the parties follow the prescribed computation.

4 Private Inference Using Garbled Circuits

Consider a model owner who holds a trained machine learning model and a client who holds a private input sample. The objective is to evaluate the model on the client's input without revealing the input to the model owner or the model parameters to the client. Following DeepSecure [2], we employ Yao's garbled-circuit (GC) protocol to perform private inference using a single cloud server.

Problem Statement. Let θ denote the private model parameters held by the model owner, and let $\mathbf{x}$ denote the private input held by the client. The parties wish to compute

$$\mathbf{y} = f_{\theta}(\mathbf{x})$$

while preserving the privacy of both $\mathbf{x}$ and θ , except for information revealed by the authorized output.

For illustration, consider a linear model

$$f_{\theta}(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b, \quad \theta = (\mathbf{w}, b).$$

The model parameters and the input are encoded as binary strings, and the inference function is represented as a Boolean circuit

$$C(\mathbf{x}, \theta) = f_{\theta}(\mathbf{x}).$$

The circuit implements the required arithmetic operations using the selected numerical representation. For real-valued inputs and parameters, fixed-point encoding can be employed, with appropriate scaling, signed arithmetic, and overflow handling.

System Model. We consider two parties:

- **Client:** A User who holds the private input $\mathbf{x}$, acts as the garbler, and retains the output decoding information.
- **Cloud server:** Holds the private model parameters θ , acts as the evaluator, and performs the garbled-circuit computation.

The circuit topology and numerical representation are assumed to be public, while the actual input and model parameters remain private. The client constructs a garbled representation $\tilde{C}$ of the circuit C without embedding the model parameters as public constants. Instead, both $\mathbf{x}$ and θ are represented as private inputs to $\tilde{C}$.

Note that the role of computing and evaluating $\tilde{C}$ can be interchanged between client and the server.

Protocol. The protocol proceeds as follows.

1. **Circuit garbling:** The client constructs $C(\mathbf{x}, \theta) = f_\theta(\mathbf{x})$, generates wire labels, and garbles the circuit to obtain $\tilde{C}$. It retains the output decoding information.
2. **Circuit transmission:** The client sends $\tilde{C}$ and its active input labels $K_{\mathbf{x}}$ to the cloud server.
3. **Model-label transfer:** The client acts as the OT sender, and the model owner acts as the receiver with choice bits representing θ . The model owner obtains the active labels K_θ through 1-out-of-2 OT and forwards them to the cloud server without disclosing the model parameters.
4. **Circuit evaluation:** The cloud evaluates the garbled circuit using $K_{\mathbf{x}}$ and K_θ :

$$K_{\mathbf{y}} = \text{Eval}(\tilde{C}, K_{\mathbf{x}}, K_\theta), \quad \mathbf{y} = f_\theta(\mathbf{x}).$$

5. **Output decoding:** The cloud returns $K_{\mathbf{y}}$ to the client, which decodes the prediction using its retained output decoding information.

Correctness. The correctness of the protocol follows from the correctness of the underlying garbling scheme. Given the active labels corresponding to $\mathbf{x}$ and θ , the server evaluates the garbled circuit to obtain the active output labels of the original circuit. Formally,

$$\text{Eval}(\tilde{C}, K_{\mathbf{x}}, K_\theta) = K_{\mathbf{y}},$$

where

$$\mathbf{y} = f_\theta(\mathbf{x}).$$

Consequently, output decoding yields

$$\text{Decode}(K_{\mathbf{y}}) = f_\theta(\mathbf{x}).$$

Security Model and Goals. We consider the semi-honest adversarial model, in which both parties follow the protocol but may attempt to infer additional information from their respective views. The security objectives are as follows:

- **Client-input privacy:** The cloud server learns no additional information about $\mathbf{x}$ beyond what is implied by its own input, the authorized output policy, and any permitted auxiliary information.
- **Model privacy:** The client learns no additional information about θ beyond what is implied by the authorized inference output, the public circuit structure, and any permitted auxiliary information.
- **Output privacy:** The server receives only the garbled output labels and does not directly obtain the decoded prediction, whereas the client is authorized to recover the output.
- **Correctness:** The client recovers the prescribed inference result $f_\theta(\mathbf{x})$.

These guarantees rely on the security of the garbling scheme and the OT protocol. The model architecture, circuit size, and numerical representation are assumed to be public. Repeated inference queries may reveal information about the model through their outputs; preventing such inference requires additional controls, such as query restrictions or output policies. The protocol described here provides semi-honest security. Malicious security requires additional mechanisms to enforce correct protocol execution.

Efficiency Optimizations. The efficiency of GC-based inference depends primarily on the Boolean circuit size, the number of non-XOR gates, and the communication required to transfer garbled tables and input labels. Techniques such as Free-XOR, half-gates, and row reduction can reduce garbling and communication costs. DeepSecure [2] further employs circuit-level optimizations and model preprocessing, including data projection and network distillation, to improve inference efficiency while controlling accuracy loss. Since the client performs garbling and the cloud server performs evaluation, the computational overhead is distributed between the two parties. The protocol requires only one cloud evaluation server and does not rely on multiple non-colluding servers.

5 Open Problem: Private Inference with a Separate Model Owner

Existing private inference protocols, including DeepSecure [2] and GAZELLE [3], protect the user’s query from the cloud server, but assume that the server knows the model parameters.

We consider a three-party setting involving a model owner, a cloud server hosting the model, and a user (querier) holding a private input $\mathbf{x}$. The objective is to compute

$$\mathbf{y} = f_{\theta}(\mathbf{x}),$$

where θ denotes the model parameters, while protecting the user’s query and preventing the cloud server from learning the model parameters. We assume that the server and user are semi-honest.

Designing a protocol that achieves both query privacy and model confidentiality in this setting remains an open problem. In particular, it is necessary to investigate whether garbled circuits can enable a single cloud server to perform inference without learning either the query or the model parameters, while providing formal security guarantees and practical efficiency.

References

- [1] A. C. Yao, “Protocols for Secure Computations,” in *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 160–164, 1982. doi: <https://doi.org/10.1109/SFCS.1982.38>.
- [2] B. Darvish Rouhani, M. Sadegh Riazi, and F. Koushanfar, “DeepSecure: Scalable Provably-Secure Deep Learning,” *arXiv preprint arXiv:1705.08963*, 2017.
- [3] A. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, “GAZELLE: A Low Latency Framework for Secure Neural Network Inference,” in *Proceedings of the 27th USENIX Security Symposium*, 2018, pp. 1651–1669.