Highly Efficient Actively Secure Two-Party Computation with One-Bit Advantage Bound

Yi Liu[1], Junzuo Lai[1], Peng Yang[2], Anjia Yang[1], Qi Wang[3], Siu-Ming Yiu[2], and Jian Weng[1]

[1] College of Cyber Security,
Jinan University, Guangzhou 510632, China
liuyi@jnu.edu.cn, laijunzuo@gmail.com, anjiayang@gmail.com, cryptjweng@gmail.com
[2] Department of Computer Science,
The University of Hong Kong, Hong Kong SAR, China
stuyangpeng@gmail.com, smyiu@cs.hku.hk
[3] Department of Computer Science and Engineering & National Center for Applied Mathematics Shenzhen,
Southern University of Science and Technology, Shenzhen 518055, China
wangqi@sustech.edu.cn

Abstract. Secure two-party computation (2PC) enables two parties to jointly evaluate a function while maintaining input privacy. Despite recent significant progress, a notable efficiency gap remains between actively secure and passively secure protocols. In S&P'12, Huang, Katz, and Evans formalized the notion of *active security with one-bit leakage*, providing a promising approach to bridging this gap. Protocols derived from this notion have become foundational in designing highly efficient actively secure 2PC protocols. However, a critical challenge identified by Huang, Katz, and Evans remains unexplored: these protocols face significant weaknesses in ensuring fairness for honest parties when employed in standalone settings rather than as components within larger protocols. While the authors proposed two potential solutions to mitigate this issue, both approaches are prohibitively expensive and lack formalization of security guarantees.

In this paper, we first formally define an enhanced notion called *active security with one-bit-advantage bound*, in which the adversaries' advantages are strictly bounded to at most one bit beyond what honest parties obtain. This bound is enforced through a *progressive revelation* mechanism, where the evaluation result is disclosed incrementally bit by bit. In addition, we propose a novel approach leveraging label structures within garbled circuits to design a highly efficient constant-round 2PC protocol that achieves active security with one-bit advantage bound. Our protocol demonstrates *runtime performance nearly identical to that of passively secure garbled-circuit counterparts* in duplex networks (*e.g.*, 1.033× for the `SHA256` circuit in LAN), with *low overhead* for output progressive revelation (only 80 communicated bytes per bit release).

With its strengthened security guarantees and minimal overhead, our protocol is highly suitable for practical 2PC applications.

1 Introduction

Secure two-party computation (2PC) [36] allows two mutually distrusting parties to jointly evaluate a function on their inputs while preserving input privacy. Based on garbled circuits [37,4], 2PC protocols for arbitrary (efficiently computable) functions can be realized in a constant number of rounds. Over the past four decades, this kind of 2PC protocols has achieved substantial improvements in both communication [2,29,24,38,32] and computation [3,15,14], leading to their broad adoption in a range of applications, either on their own or as fundamental building blocks.

Despite these advancements, a *significant efficiency gap* remains between actively secure (a.k.a. malicious) and passively secure (a.k.a. semi-honest) 2PC protocols. From the perspective of communication, the size of garbled circuits for passive security settings recently has been reduced from 2κ bits per AND gate (using the half-gates scheme [38]) to $1.5\kappa + 5$ bits (using the three-halves scheme [32]). However, even with recent progress in constant-round actively secure 2PC [34,22,35,10,7], the latest protocol [7] has only reached $2\kappa + 5$ bits per AND gate for one-way communication.[4] From a

[4] The two-way communication cost of this protocol is $4\kappa + 10$ bits per AND gate. In duplex networks, we can only consider one-way communication cost. Though approaches like the GMW compiler [12,1] could potentially close this communication gap, their computational overheads are prohibitively high.

computational standpoint, recent actively secure protocols [10,7] are notably complex and thus *no* implementations are currently available. While they theoretically achieve communication overhead comparable with passively secure 2PC using the earlier half-gates scheme, it remains to be experimentally verified whether the overall runtime benefits from the added complicated components in the protocol, as many of these introduce significant overhead. Existing implementations [34,18,35] demonstrate that passively secure 2PC protocols are still at least an order of magnitude faster than actively secure protocols for small circuits (*e.g.*, `SHA-256`). This gap is likely to be widened even further for larger circuits, where components such as oblivious transfer (OT) account for a smaller fraction of total runtime.

The notion of *active security with one-bit leakage* [27,19,23,17,26,39] offers a promising approach to bridging the efficiency gap between passive and active security in constant-round 2PC. Originally introduced by Mohassel and Franklin [27] and later formalized by Huang, Katz, and Evans [19], this security model guarantees that *the adversary is limited to learning at most one additional bit of information about the honest party's input beyond the evaluation result, while output correctness is preserved*. Huang, Katz, and Evans developed a framework for designing actively secure 2PC protocols with one-bit leakage based on garbled circuits, referred to in this paper as *the HKE framework*. This is the most common framework for designing one-bit-leakage protocols. By applying this framework to the state-of-the-art garbling scheme, we could obtain a *highly efficient* actively secure 2PC protocol with one-bit leakage, with runtime *very close* to that of its passively secure counterpart. Thus, although a single bit of extra information may be revealed to the adversary, this trade-off provides significant efficiency gains. In recent years, due to the high efficiency and security guarantees, protocols derived from the HKE framework have been *widely adopted as a component* for designing actively secure 2PC [28,23,30,7,39], 2PC with robust publicly verifiable covert security [26], and private set intersection (PSI) [31].

While one-bit-leakage protocols derived from the HKE framework are highly efficient and can limit the adversary to gain at most one extra bit of information, when these protocols are used in *standalone settings* instead of being used as a component in other protocols, they are *practically limited*. This limitation arises because, when used on their own, these protocols have critical security weaknesses in fairness for honest parties. An adversary in one-bit-leakage protocols can *always* choose to learn the evaluation result, together with an extra bit of information about the honest party's inputs, while having the honest party learn *nothing*. Specifically, in these protocols, parties will obtain an evaluation result (which may be incorrect if misbehavior occurs) and then perform a verification to confirm its correctness. As a result, an honest party might initially obtain an incorrect result, only to later reject it upon verification, ultimately learning nothing (see Section 2.2 for more details). This advantage for adversaries actually encourage dishonest behaviors, which is practically unacceptable in many scenarios, thereby severely limiting the direct applicability of one-bit-leakage 2PC.

Huang, Katz, and Evans indeed identified this critical problem in their original work [19] and proposed two heuristic enhancements to address it. The first approach aims to verify the garbled outputs, *i.e.*, (potentially incorrect) evaluation results in encrypted form, before revealing them, thus preventing an adversary from learning the result if they are caught cheating during verification. To accomplish this, they employ a fully-fledged actively secure garbled circuit protocol that takes the garbled outputs as inputs for verification. However, this approach is *prohibitively expensive*. For example, when evaluating an `AES-128` circuit with a (2×128)-bit input and a 128-bit output, the garbled outputs comprise 128 output-wire labels, each 128 bits in length. These 128×128 bits would serve as a portion of inputs for the verification's garbled circuit. In fact, this approach requires a complex verification circuit with at least 6×128^2 bits of inputs. Such an approach is even more expensive than evaluating `AES-128` directly with fully-fledged active security. The second enhancement is progressive revelation [5,11,21,8]. It ensures that the adversary's advantage is limited to only one additional bit of output beyond what the honest party obtains, by revealing the evaluation result bit by bit. However, this also relies on a fully-fledged actively secure garbled circuit protocol to process output-wire labels from the one-bit-leakage protocol, resulting in significant overhead. These approaches, while theoretically addressing the fairness problem, *impose expensive overhead and conflict with the original goal of developing efficient actively secure protocols*. Additionally, *no clear method to combine these approaches has been provided*, and *what security guarantees they provide lacks formalization*.

Therefore, the following question is open so far:

How can we provide an effective solution to the fairness problem in one-bit-leakage protocols?

1.1 Our Contributions

In this paper, we integrate both of the aforementioned enhancements within a new formalized security model and provide a *highly efficient* protocol that effectively addresses this fairness problem. Specifically:

New security notion. We propose a new notion for 2PC named *active security with one-bit advantage bound* to address the fairness problem inherent in one-bit leakage protocols. Protocols achieving active security with one-bit advantage bound limit the *advantage* of an (actively) corrupted party to at most one bit. That is, instead of $n+1$ bits in one-bit leakage protocols, *the adversary can gain at most one more bit of information beyond what the honest party obtains.* The security model employs a *progressive revelation* mechanism in which one bit of the evaluation result is disclosed iteratively. It also supports *adjustable advantage*, allowing the adversary's one-bit advantage to be adjusted to arbitrary k-bit advantage.[5]

Protocol with low overhead. We design a *highly efficient* constant-round 2PC protocol that achieves active security with one-bit advantage bound. Note that our protocol is *fully compatible* with the state-of-the-art garbling schemes – the three-halves scheme and the half-gates scheme.[6] Besides the progressive revelation phase, the runtime of our protocol is *almost identical* to that of the passively secure counterpart in duplex networks (*e.g.*, 1.033× for the `SHA256` circuit in LAN, see more in Section 6). The progressive revelation phase itself is highly efficient: the number of rounds can be minimized to almost the same as the bit-length of the evaluation result, while the one-way communication required to securely release each bit of the result is only 80 bytes, with no dependency on public-key operations.

With its stronger security guarantees and low overhead, our improved protocol is highly suitable for practical applications of secure two-party computation.

2 Technical Overview

In this section, we first introduce the notations used throughout the paper, and then present an overview of the HKE framework for designing actively secure 2PC protocols with *one-bit-leakage* and describe why the aforementioned fairness problem arises. Subsequently, we provide intuition behind our improved 2PC protocol, detailing how the structure of wire labels in garbling schemes can be exploited to achieve active security with *one-bit advantage bound*.

2.1 Notation

We denote the security parameter by κ, which may be used as an implicit input to algorithms in this paper. We denote the size of a set S as $|S|$ and use the notation $x \leftarrow\!\!\$\, S$ to indicate that an element x is sampled uniformly at random from S. For any positive integer n, let $[n] = \{1, \ldots, n\}$, and for positive integers $a < b$, define $[a, b] = \{a, a+1, a+2, \ldots, b\}$. For a vector $\boldsymbol{v}$, we denote the ith element of $\boldsymbol{v}$ as $\boldsymbol{v}[i]$. For a string $s \in \{0,1\}^*$, the ith bit of s is written as either s_i or $s[i]$.

We write $\mathbb{F}_{2^\kappa} \cong \mathbb{F}_2[X]/f(X)$ for monic irreducible polynomial $f(X)$ of degree κ and use $\mathsf{X} \in \mathbb{F}_{2^\kappa}$ to represent the element corresponding to $X \in \mathbb{F}_2[X]/f(X)$. When it is clear from the context, we interchangeably use $\{0,1\}^\kappa$, $\mathbb{F}_2^\kappa$, and $\mathbb{F}_{2^\kappa}$. Hence, addition in $\mathbb{F}_2^\kappa$ and $\mathbb{F}_{2^\kappa}$ corresponds to XOR in $\{0,1\}^\kappa$.

A Boolean circuit $\mathcal{C}$ consists of a list of gates, where each gate is given in the form of (i, j, k, T). Here, i and j are the indices of input wires, k is the index of output wire, and $T \in \{\oplus, \wedge\}$ specifies the gate type. We denote the set of circuit wire indices as $\mathcal{W}$, with n representing the number of input wires and n_{O} the number of output wires. For the ith output wire, the function out maps $i \in [n_{\mathsf{O}}]$ to a wire index $w \in \mathcal{W}$. The set of output wires is represented as $\mathcal{W}_{\mathsf{O}} = \{w \mid w = \mathsf{out}(i) \text{ for } i \in [n_{\mathsf{O}}]\}$, with $n_{\mathsf{O}} = |\mathcal{W}_{\mathsf{O}}|$.

In the 2PC setting, let $\mathcal{W}_{\mathsf{A}}$ and $\mathcal{W}_{\mathsf{B}}$ represent the sets of indices for P_{A}'s and P_{B}'s input wires, respectively, where $n_{\mathsf{A}} = |\mathcal{W}_{\mathsf{A}}|$ and $n_{\mathsf{B}} = |\mathcal{W}_{\mathsf{B}}|$, satisfying $n = n_{\mathsf{A}} + n_{\mathsf{B}}$.

[5] This can reduce the number of rounds and is useful for scenarios where the length of evaluation result is long, or the network latency is high, allowing trade-off between security and efficiency.

[6] Recent implementations of the three-halves scheme [16,6] demonstrate that the half-gates scheme outperforms the three-halves scheme in many common scenarios. Accordingly, we consider both schemes to represent the state-of-the-art.

2.2 Overview of Prior Work

The HKE framework employs a technique called dual execution to limit the leakage to one bit. The core idea of dual execution is that two parties $\mathsf{P_A}$ and $\mathsf{P_B}$ execute the classical garbled circuit protocol twice over the same circuit and inputs, where actively secure OT protocols for input-wire retrieval are used. In one execution, $\mathsf{P_A}$ acts as the garbler and $\mathsf{P_B}$ as the evaluator; in the other, their roles are reversed, *i.e.*, $\mathsf{P_B}$ becomes the garbler, and $\mathsf{P_A}$ becomes the evaluator. Each party also sends the decoding information for their garbled outputs to the other party. Once both executions are complete, the parties perform an actively secure *equality test* on the output materials of the two garbled circuits to determine the final result. Specifically, for an n_{O}-bit-output circuit, the honest $\mathsf{P_A}$'s input to the equality test includes output-wire labels $\left\{Y'_i, Y_{i,y'_i}\right\}_{i\in[n_{\mathsf{O}}]}$, where Y'_i (resp. y'_i) denotes the ith output-wire label (resp. the ith bit of output) of the garbled circuit generated by $\mathsf{P_B}$ and evaluated by $\mathsf{P_A}$, while Y_{i,y'_i} represents the ith output-wire label generated by $\mathsf{P_A}$ with real bit value y'_i. The inputs provided by $\mathsf{P_B}$ to the equality test follow a similar structure. If the test passes, both parties accept the output as correct; otherwise, the protocol terminates, signaling that one party is corrupted and has deviated from the protocol. Given that actively secure OTs are cheap, the total cost is only around twice as much as a passively secure garbled circuit protocol, and it is nearly no overhead in duplex networks.

This method effectively limits a corrupted party to learning, at most, a single additional bit of information about the honest party's input beyond the evaluation result. The reasoning is that while a corrupted party may deviate from the protocol when garbling the circuit, it cannot cheat in the circuit evaluation as the evaluator if the OT protocols are actively secure. Therefore, the only additional information the corrupted party can gain from the incorrectly generated garbled circuit is the output of the equality test — either a true or false result, providing at most one bit of information that could potentially depend on the honest party's input.

Nevertheless, as we have mentioned in Section 1, the security guarantees offered by the one-bit-leakage protocols derived from HKE framework are still practically limited due to the fairness problem. In particular, for the equality test, $\mathsf{P_A}$ must obtain the evaluation result y' of $\mathsf{P_B}$'s garbled circuit beforehand (from the decoding information). Therefore, a corrupted $\mathsf{P_B}$ can *always* learn the evaluation result by evaluating $\mathsf{P_A}$'s garbled circuit while generating a malicious circuit for $\mathsf{P_A}$, thereby gaining an extra bit of information about $\mathsf{P_A}$'s inputs through the equality test. As a result, a corrupted $\mathsf{P_B}$ could *always* obtain $n+1$ bits of information while having the honest $\mathsf{P_A}$ learn *nothing*.

2.3 Overview of Our Solution

We introduce the idea behind our 2PC protocol in the following. Section 4.2 formally describes our protocol in detail, and some improvement techniques and optimizations that could be used in implementations are introduced in Section 5.

In protocols derived from the HKE framework, the parties may obtain a potentially incorrect evaluation result from the garbled circuit generated by the other party, which they need to verify through an equality test later. To solve the aforementioned fairness problem, it is necessary to verify the correctness of the output once it has been released or even before it is released. In our protocol, we maintain the dual execution procedure as previously described, with one key modification: neither party transmits the decoding information for garbled outputs to the other party. This change delays the point at which both parties learn the evaluation result, and both parties, as the evaluator, only hold the garbled outputs after the garbled circuit evaluation. From this stage onward, we divide the protocol into two main phases: verification and progressive revelation.

In the verification phase, we introduce a novel approach that enables both parties to *blindly* verify, via one equality test, whether the garbled outputs from their respective circuits encode the same evaluation result without disclosing it. If this test fails, the protocol terminates, and the corrupted party learns no more than one bit of information from the test outcome. Conversely, if the test passes, it confirms that even if the corrupted party may have improperly generated the garbled circuit, both parties still hold the consistent evaluation result. Intuitively, if the underlying OT protocol is actively secure, the evaluator cannot deviate undetected within the garbled circuit framework. Thus, the successful completion of this test assures the honest party that the evaluation result of the other party's garbled circuit aligns with the evaluation result of his own garbled circuit, thereby preserving the *correctness* of the evaluation result.

The protocol then advances to the progressive revelation phase, where the evaluation result is gradually disclosed, *i.e.*, output in a bit-by-bit manner. During this phase, each bit of the evaluation result is sequentially opened and verified. If any bit fails to be opened correctly, the protocol aborts, limiting the adversary's gain to at most one additional bit of information. Moreover, this gradual opening procedure, bounded by a one-bit advantage, can be adapted to release k bits of the evaluation result at a time, thereby reducing the protocol's round complexity through a security trade-off.

We begin by describing our method for verifying that the garbled outputs from both parties encode the same evaluation result without disclosing it. This method is compatible with the current state-of-the-art half-gates and three-halves garbling schemes. To set the stage, we first briefly review the label structure utilized in these garbling schemes, and then introduce how to exploit this structure to achieve our goal.

Both the half-gates and three-halves garbling schemes employ the point-and-permute and free-XOR techniques. For the point-and-permute technique, the actual bit z_w on each wire w is masked by a point-permute bit (a.k.a. random bit) λ_w generated by the garbler, and the resulting masked bit $\hat{z}_w = z_w \oplus \lambda_w \in \{0,1\}$ allows to be known by the evaluator for garbled circuit evaluation. The free-XOR technique enables XOR gates to be garbled with no communication. In particular, the garbler sets $\mathsf{L}_{w,0} \oplus \mathsf{L}_{w,1} = \Delta$ for each wire w, where $\mathsf{L}_{w,b} \in \{0,1\}^\kappa$ is the label for wire w with masked bit $b \in \{0,1\}$, and $\Delta \in \{0,1\}^\kappa$ is a global key (a.k.a. fixed offset). The output-wire label of an XOR gate is computed by simply XORing the two input-wire labels of that gate. The masked bit is often encoded in the least significant bit of the wire labels, and thereby let $\mathsf{lsb}(\Delta) = 1$ resulting in $\hat{z}_w = \mathsf{lsb}(\mathsf{L}_{w,\hat{z}_w})$.

Suppose that a garbler $\mathsf{P_A}$ and an evaluator $\mathsf{P_B}$ leverage a garbling scheme with point-and-permute and free-XOR technique (*e.g.*, half-gates or three-halves) to execute a passively secure 2PC protocol for a circuit $\mathcal{C}$, where only $\mathsf{P_A}$ obtains the final n-bit outputs. $\mathsf{P_A}$ generates and sends a garbled circuit to $\mathsf{P_B}$. Both parties then execute an OT protocol, such that $\mathsf{P_B}$ obliviously retrieves the input-wire labels corresponding to his inputs from $\mathsf{P_A}$. $\mathsf{P_A}$ also sends input-wire labels corresponding to her inputs to $\mathsf{P_B}$. Given all input-wire labels, $\mathsf{P_B}$ can evaluate the garbled circuit.

Denote the global key by Δ_A. During the evaluation of garbled circuit, for a wire w, the evaluator $\mathsf{P_B}$ holds the wire label $\mathsf{L}_{w,\hat{z}_w} = \mathsf{L}_{w,0} \oplus \hat{z}_w \Delta_\mathsf{A}$, which also contains the masked bit $\hat{z}_w = \mathsf{lsb}(\mathsf{L}_{w,\hat{z}_w})$. Meanwhile, the garbler $\mathsf{P_A}$ holds the point-permute bit λ_w. In other words, the actual bit $z_w = \hat{z}_w \oplus \lambda_w$ remains secret and is shared between the two parties.

After the evaluation, the evaluator $\mathsf{P_B}$ sends $\mathsf{L}_{w,\hat{z}_w}$ (implicitly containing $\hat{z}_w$) for each output wire $w \in \mathcal{W}_\mathsf{O}$ to the garbler $\mathsf{P_A}$. The garbler $\mathsf{P_A}$ then computes $z_w = \hat{z}_w \oplus \lambda_w$ and verifies whether $\mathsf{L}_{w,\hat{z}_w} = \mathsf{L}_{w,0} \oplus (\lambda_w \oplus z_w)\Delta_\mathsf{A}$. This verification is equivalent to checking whether the following holds.

$$\mathsf{L}_{w,\hat{z}_w} \oplus (\mathsf{L}_{w,0} \oplus \lambda_w \Delta_\mathsf{A}) = z_w \Delta_\mathsf{A} \,.$$

By the authenticity property of garbling schemes, a corrupted evaluator $\mathsf{P_B}$ cannot send a tampered label $\hat{\mathsf{L}}_w \neq \mathsf{L}_{w,\hat{z}_w}$ to pass this check, except with negligible probability. Notably, before $\mathsf{P_B}$ sends $\mathsf{L}_{w,\hat{z}_w}$, $\mathsf{P_B}$ holds $\mathsf{L}_{w,\hat{z}_w} \in \{0,1\}^\kappa$, while $\mathsf{P_A}$ can locally compute $\mathsf{L}_{w,0} \oplus \lambda_w \Delta_\mathsf{A} \in \{0,1\}^\kappa$. Therefore, $\mathsf{P_A}$ and $\mathsf{P_B}$ hold secret shares of both $z_w \in \{0,1\}$ and $z_w \Delta_\mathsf{A} \in \{0,1\}^\kappa$.

Similarly, if the roles of the parties are reversed, with $\mathsf{P_B}$ as the garbler and $\mathsf{P_A}$ as the evaluator for the same circuit $\mathcal{C}$, then for an output wire w, $\mathsf{P_A}$ holds the masked bit $\hat{z}'_w$ and the wire label $\mathsf{L}'_{w,\hat{z}'_w}$. Meanwhile, $\mathsf{P_B}$ holds λ'_w and can compute $\mathsf{L}'_{w,0} \oplus \lambda'_w \Delta_\mathsf{B}$. Thus, in this case, $\mathsf{P_A}$ and $\mathsf{P_B}$ secretly share z'_w and $z'_w \Delta_\mathsf{B}$.

The table below summarizes the values held by $\mathsf{P_A}$ and $\mathsf{P_B}$ for wire w during the *dual* executions, where the garbled circuit GC is generated by $\mathsf{P_A}$ while GC' is generated by $\mathsf{P_B}$.

	Values held by $\mathsf{P_A}$	Values held by $\mathsf{P_B}$
GC	Δ_A, $\mathsf{L}_{w,0}$, λ_w	$\hat{z}_w$, $\mathsf{L}_{w,\hat{z}_w} = \mathsf{L}_{w,0} \oplus \hat{z}_w \Delta_\mathsf{A}$
GC'	$\hat{z}'_w$, $\mathsf{L}'_{w,\hat{z}'_w} = \mathsf{L}'_{w,0} \oplus \hat{z}'_w \Delta_\mathsf{B}$	Δ_B, $\mathsf{L}'_{w,0}$, λ'_w

For $i \in [n_\mathsf{O}]$ and $w := \mathsf{out}(i)$, define $B_{i,1} := \mathsf{L}_{w,\hat{z}_w}$ (held by $\mathsf{P_B}$), $A_{i,1} := \mathsf{L}_{w,0} \oplus \lambda_w \Delta_\mathsf{A}$ (held by $\mathsf{P_A}$), and $\omega_i = z_w$. We have

$$A_{i,1} \oplus B_{i,1} = (\lambda_w \oplus \hat{z}_w)\Delta_\mathsf{A} = z_w \Delta_\mathsf{A} = \omega_i \Delta_\mathsf{A} \,.$$

Similarly, let $A_{i,2} := \mathsf{L}'_{w,\hat{z}'_w}$ (held by $\mathsf{P_A}$), $B_{i,2} := \mathsf{L}'_{w,0} \oplus \lambda'_w \Delta_\mathsf{B}$ (held by $\mathsf{P_B}$), and $\omega'_i = z'_w$. We have

$$A_{i,2} \oplus B_{i,2} = (\lambda'_w \oplus \hat{z}'_w)\Delta_\mathsf{B} = z'_w \Delta_\mathsf{B} = \omega'_i \Delta_\mathsf{B} \,.$$

Now $\mathsf{P_A}$ can compute $A_i := A_{i,1} \oplus A_{i,2}$, while $\mathsf{P_B}$ can compute $B_i := B_{i,1} \oplus B_{i,2}$. Let $\Delta = \Delta_\mathsf{A} \oplus \Delta_\mathsf{B}$. If both parties are honest, we expect $\omega_i = z_w = \hat{z}_w \oplus \lambda_w = \hat{z}'_w \oplus \lambda'_w = z'_w = \omega'_i$, and thus

$$\begin{aligned} A_i \oplus B_i &= A_{i,1} \oplus A_{i,2} \oplus B_{i,1} \oplus B_{i,2} = \omega_i \Delta_\mathsf{A} \oplus \omega'_i \Delta_\mathsf{B} \\ &= \omega_i \Delta = \omega'_i \Delta \,. \end{aligned}$$

In this scenario, $\mathsf{P_A}$ and $\mathsf{P_B}$ effectively share the values ω_i and $\omega_i \Delta$. Although derived from different methods, this resembles the SPDZ authentication, where both parties *secretly share* the global key Δ, the value $\omega_i = \omega'_i$, and the corresponding message authentication code (MAC) $\omega_i \Delta$.

If one party is corrupted, the outputs of the two garbled circuits from dual executions may not be consistent. Our key insight for verifying the consistency of garbled outputs stems from the technique of opening SPDZ-style authenticated secret sharing.

Without loss of generality, assume that $\mathsf{P_A}$ is honest and $\mathsf{P_B}$ is corrupted. In the case of garbled circuits, if the OT protocol is actively secure, a corrupted party cannot cheat (without being detected) when playing the role of the evaluator due to the authenticity property of garbling schemes. In the execution where the corrupted $\mathsf{P_B}$ is the garbler, the honest $\mathsf{P_A}$ having input x_A is indeed evaluating a function F^*_B that could be fully defined by $\mathsf{P_B}$:

$$\left\{ (\hat{z}'_w, \mathsf{L}'_{w,\hat{z}'_w}) \right\}_{w \in \mathcal{W}_\mathsf{O}} := F^*_\mathsf{B}(x_\mathsf{A}) \,.$$

As previously mentioned, for an output wire w, the two parties hold a SPDZ-like authenticated secret sharing of the actual value. They can open this bit as $\omega''_i := \hat{z}'_w \oplus \lambda'_w$,[7] where $i = \mathsf{out}^{-1}(w)$, and then proceed with an SPDZ-style opening verification for ω''_i. In this SPDZ-style verification, $\mathsf{P_A}$ and $\mathsf{P_B}$ must commit to and subsequently open the values $A_i \oplus \omega''_i \Delta_\mathsf{A}$ and $B_i \oplus \omega''_i \Delta_\mathsf{B}$, respectively. The verification checks whether these two committed values are equal. Specifically, $\mathsf{P_A}$ will commit to and open the value:

$$\begin{aligned} A_i \oplus \omega''_i \Delta_\mathsf{A} &= A_{i,1} \oplus A_{i,2} \oplus \omega''_i \Delta_\mathsf{A} \\ &= \mathsf{L}_{w,0} \oplus \lambda_w \Delta_\mathsf{A} \oplus A_{i,2} \oplus \omega''_i \Delta_\mathsf{A} \,, \end{aligned}$$

where $A_{i,2} = \mathsf{L}'_{w,\hat{z}'_w}$ is derived from F^*_B. To pass this verification, $\mathsf{P_B}$ must also commit to and open a value that equals $A_i \oplus \omega''_i \Delta_\mathsf{A}$ (*i.e.*, the XOR of the two committed values must be zero). It is important to note that the value held by the corrupted $\mathsf{P_B}$ corresponding to $\mathsf{L}_{w,0}$ is $B_{i,1} = \mathsf{L}_{w,\hat{z}_w} = \mathsf{L}_{w,0} \oplus \hat{z}_w \Delta_\mathsf{A}$. Thus, to cancel out $\mathsf{L}_{w,0}$, $\mathsf{P_B}$'s committed value must include $B_{i,1}$ (in the form of XOR), except with negligible probability of simple guessing. Given that $B_{i,1} \oplus A_i \oplus \omega''_i \Delta_\mathsf{A} = z_w \Delta_\mathsf{A} \oplus A_{i,2} \oplus \omega''_i \Delta_\mathsf{A} = A_{i,2} \oplus (z_w \oplus \omega''_i) \Delta_\mathsf{A}$, we observe that if $z_w \oplus \omega''_i \neq 0$, *i.e.*, the opened bit ω''_i is inconsistent with the output of the garbled circuit generated by the honest party $\mathsf{P_A}$, then $\mathsf{P_B}$ can only make his committed value match $\mathsf{P_A}$'s by correctly guessing the value of Δ_A, which occurs with negligible probability, even though $\mathsf{P_B}$ has full control over the value $A_{i,2}$. A similar conclusion holds if $\mathsf{P_A}$ is corrupted and $\mathsf{P_B}$ is honest. Therefore, except with negligible probability, this procedure allows the two parties to determine whether the opened bit is correct.

Can this procedure be *directly* applied to open and verify the output of garbled circuits? *The answer is no.* For example, a corrupted $\mathsf{P_B}$ could manipulate F^*_B so that the honest $\mathsf{P_A}$ obtains $A_{i,2} = 0$. Since $B_{i,1} \oplus A_i \oplus \omega''_i \Delta_\mathsf{A} = A_{i,2} \oplus (z_w \oplus \omega''_i) \Delta_\mathsf{A} = (z_w \oplus \omega''_i) \Delta_\mathsf{A}$, when $z_w \oplus \omega''_i = 1$, after observing $\mathsf{P_A}$'s committed value $A_i \oplus \omega''_i \Delta_\mathsf{A}$, $\mathsf{P_B}$ can can deduce the value of Δ_A. Even though the protocol would terminate due to an invalid opening, $\mathsf{P_B}$ could still exploit Δ_A to open the previously received garbled circuit generated by $\mathsf{P_A}$ and learn $\mathsf{P_A}$'s inputs. This problem can be resolved by having both parties execute a secure *equality test* protocol instead, which only reveals whether the two committed values are identical.

This verification can be done in batches. We can treat bits and κ-bit strings as elements in $\mathbb{F}_2^\kappa$ and $\mathbb{F}_{2^\kappa}$. To batch-verify the opened bits, both parties can first use coin-tossing protocols to select random values $r_i \leftarrow_\$ \mathbb{F}_2^\kappa$ for $i \in [n_\mathsf{O}]$ and compute $\omega'' = \sum_{i \in [n_\mathsf{O}]} r_i \omega''_i \in \mathbb{F}_2^\kappa$. Then $\mathsf{P_A}$ and $\mathsf{P_B}$ compute $A := \sum_{i \in [n_\mathsf{O}]} r_i A_i$ and $B := \sum_{i \in [n_\mathsf{O}]} r_i B_i$, respectively, such that $A \oplus B = \omega'' \Delta$. To verify ω'', $\mathsf{P_A}$

[7] Reconstructing this bit as $\omega''_i := \hat{z}_w \oplus \lambda_w$ using shares from a garbled circuit generated by the honest party $\mathsf{P_A}$ is also an option. However, here we only aim to introduce the intuition behind our protocol and do not concern ourselves with how ω'' is reconstructed.

computes $A \oplus \omega'' \Delta_{\mathsf{A}}$, where

$$\begin{aligned} A \oplus \omega'' \Delta_{\mathsf{A}} &= \left(\sum_{i \in [n_{\mathsf{O}}]} r_i A_i \right) \oplus \left(\sum_{i \in [n_{\mathsf{O}}]} r_i \omega_i'' \right) \Delta_{\mathsf{A}} \\ &= \sum_{i \in [n_{\mathsf{O}}]} r_i (A_i \oplus \omega_i'' \Delta_{\mathsf{A}}) \\ &= \sum_{i \in [n_{\mathsf{O}}]} r_i (\mathsf{L}_{w,0} \oplus \lambda_w \Delta_{\mathsf{A}} \oplus A_{i,2} \oplus \omega_i'' \Delta_{\mathsf{A}}) \,, \end{aligned}$$

and uses this in the equality test. To make these openings valid, corrupted P_{B} must provide a value equal to $A \oplus \omega'' \Delta_{\mathsf{A}}$. Since r_i's are randomly chosen and $|\mathbb{F}_2^\kappa|$ is large enough, if the equality test passes, then $A_i \oplus \omega_i'' \Delta_{\mathsf{A}}$ is equal to $B_i \oplus \omega_i'' \Delta_{\mathsf{B}}$ except with negligible probability. Similar to verifying a single opening, in the equality test, P_{B}'s value must account for all $r_i B_{i,1}$ terms (in XOR form) to cancel out $\mathsf{L}_{w,0}$, leaving only a negligible probability of successful guessing. Then, if $z_w \neq \omega_i''$, the only way for P_{B} to align their values with those of P_{A} is by correctly guessing Δ_{A}.

Since both parties hold authenticated secret shares of each ω_i'' for $i \in [n_{\mathsf{O}}]$, they can locally compute the authenticated secret sharing of ω'' as a whole, opening only ω'' rather than revealing each individual ω_i''. This process is equivalent to verifying that, for each pair of MAC shares (A_i, B_i), the two parties correctly possess correct shares of the corresponding bit. In the dual execution setting, these shared bits appear as $\omega_i = \hat{z}_w \oplus \lambda_w$ or/and $\omega_i' = \hat{z}_w' \oplus \lambda_w'$. Therefore, both ω_i's and ω_i''s can be verified simultaneously in batch. To accomplish this, the parties jointly sample random values $r_i, r_i' \leftarrow\!\!\$\, \mathbb{F}_2^\kappa$ and locally compute the share of $\omega := \left(\sum_{i \in [n_{\mathsf{O}}]} r_i \omega_i\right) \oplus \left(\sum_{i \in [n_{\mathsf{O}}]} r_i' \omega_i'\right)$, which is then opened. They subsequently compute the associated MACs, defined as $A := \left(\sum_{i \in [n_{\mathsf{O}}]} r_i A_i\right) \oplus \left(\sum_{i \in [n_{\mathsf{O}}]} r_i' A_i\right)$ and $B := \left(\sum_{i \in [n_{\mathsf{O}}]} r_i B_i\right) \oplus \left(\sum_{i \in [n_{\mathsf{O}}]} r_i' B_i\right)$, using these to validate ω.

Now, with both parties holding MAC shares $\{(A_i, B_i)\}_{i \in [n_{\mathsf{O}}]}$, they can efficiently verify whether their shares satisfy the condition $\omega_i = \hat{z}_w \oplus \lambda_w = \hat{z}_w' \oplus \lambda_w' = \omega_w'$ for $w = \mathsf{out}(i)$ and $i \in [n_{\mathsf{O}}]$, ensuring both correctness and authentication by revealing only ω and performing a single equality test. This approach eliminates the need to individually open ω_i and ω_i'. However, a direct revelation of ω would expose a linear combination of ω_i and ω_i'. To conceal this linear combination result, the two parties can generate an authenticated secret sharing of a random value $\omega_0 \leftarrow\!\!\$\, \mathbb{F}_2^\kappa$ and incorporate it into the linear combination. Let A_0 and B_0 represent the MACs held by P_{A} and P_{B}, respectively, where $A_0 \oplus B_0 = \omega_0 \cdot \Delta$. The two parties can then open $\omega := \omega_0 \oplus \left(\sum_{i \in [n_{\mathsf{O}}]} r_i \omega_i\right) \oplus \left(\sum_{i \in [n_{\mathsf{O}}]} r_i' \omega_i'\right)$ and verify its correctness as previously outlined. By introducing the random value ω_0, this approach effectively conceals the linear combination of the shares.

At this stage, both parties have *blindly* verified the consistency of the outputs from the garbled circuits generated by P_{A} and P_{B} through a single equality test. If the outputs are consistent, they proceed to the progressive revelation phase. With both parties holding $\omega_i = \omega_i'$ for all $i \in [n_{\mathsf{O}}]$, they can gradually open each bit of the evaluation result and verify its correctness. Specifically, P_{A} sends $a_{i,1} = \lambda_w$ to P_{B} while P_{B} sends $b_{i,2} = \lambda_w'$ to P_{A} for $w = \mathsf{out}(i)$. Then P_{A} and P_{B} can reconstruct $\omega_i' = a_{i,2} \oplus b_{i,2}$ and $\omega_i = a_{i,1} \oplus b_{i,1}$, respectively. An equality test is subsequently used to verify the opening of each bit with respect to A_i and B_i: P_{A} inputs $A_i \oplus \omega_i' \Delta \mathsf{A}$ and P_{B} inputs $B_i \oplus \omega_i \Delta_{\mathsf{B}}$. If any verification fails, the protocol terminates; otherwise, both parties securely obtain the final evaluation result. Throughout these two phases, a corrupted party gains at most a one-bit advantage over the honest party, yielding an actively secure 2PC protocol with a one-bit advantage bound.

Additionally, by batching the opening and verification process of the progressive revelation phase through random linear combinations, k bits of the evaluation result can be opened and verified simultaneously. This approach yields an actively secure 2PC protocol with a k-bit advantage bound, thereby reducing round complexity. We note that k could even be different in each iteration to adapt to different scenarios flexibly.

3 One-Bit Advantage Bound

Based on the discussion in Section 2, we define the ideal functionality $\mathcal{F}_{\mathsf{OneBitAdv}}$ for 2PC with a one-bit advantage bound as follows. A two-party protocol $\Pi_{\mathsf{OneBitAdv}}$ for the circuit $\mathcal{C}$ is said to securely compute $\mathcal{C}$ with one-bit advantage bound if it securely realizes the functionality $\mathcal{F}_{\mathsf{OneBitAdv}}$.

Functionality $\mathcal{F}_{\mathsf{OneBitAdv}}$

Public inputs: Both parties agree on a circuit $\mathcal{C}$ with $n = n_{\mathsf{A}} + n_{\mathsf{B}}$ input wires and n_{O} output wires.
Private inputs: P_{A} has input $x_{\mathsf{A}} \in \{0,1\}^{n_{\mathsf{A}}}$, whereas P_{B} has input $x_{\mathsf{B}} \in \{0,1\}^{n_{\mathsf{B}}}$.

Upon receiving x_{A} from P_{A} and x_{B} from P_{B}, store these values and compute $y := \mathcal{C}(x_{\mathsf{A}}, x_{\mathsf{B}})$.

Verification

Upon receiving a set of functions $\{g_i, g_i'\}_{i\in[n_{\mathsf{O}}]}$, where $g_i, g_i' \colon \{0,1\}^n \to \mathbb{F}_2^\kappa$, from the adversary, compute $\alpha_i := g_i(x_{\mathsf{A}}, x_{\mathsf{B}})$ and $\alpha_i' := g_i'(x_{\mathsf{A}}, x_{\mathsf{B}})$ for each $i \in [n_{\mathsf{O}}]$. Then, send random $r_i, r_i' \leftarrow\!\!\$\, \mathbb{F}_2^\kappa$ for $i \in [n_{\mathsf{O}}]$ to the adversary and receive $\beta \in \mathbb{F}_2^\kappa$ in return.
Verify whether $\beta = \left(\sum_{i\in[n_{\mathsf{O}}]} r_i\alpha_i\right) \oplus \left(\sum_{i\in[n_{\mathsf{O}}]} r_i'\alpha_i'\right)$. Send the verification result to the adversary. If the verification fails, send $\mathsf{cheating}$ to the honest party and terminate. Otherwise, send $\{(\alpha_i, \alpha_i')\}_{i\in[n_{\mathsf{O}}]}$ to the adversary and proceed.

Progressive revelation

For each $i \in [n_{\mathsf{O}}]$:

- Send y_i to the adversary.
- If $\mathsf{continue}$ from the adversary is received, send y_i to the honest party and continue. If abort from the adversary is received, send $\perp$ to the honest party and terminate.

We note that the ideal functionality $\mathcal{F}_{\mathsf{OneBitAdv}}$ borrows the idea for the definition of one-bit leakage [19]. In the definition of one-bit leakage, the adversary in the ideal world can send a Boolean function g to the ideal functionality and receive the one-bit outcome of g applied to both parties' inputs, thus defining the one-bit leakage. To better align with our security proof, we adapt this approach. Instead, during the verification phase, the adversary is allowed to submit a set of functions $\{g_i, g_i'\}$ to $\mathcal{F}_{\mathsf{OneBitAdv}}$. The functionality responds by generating random values r_i and r_i' for each $i \in [n_{\mathsf{O}}]$, while computing $\alpha_i := g_i(x_{\mathsf{A}}, x_{\mathsf{B}})$ and $\alpha_i' := g_i'(x_{\mathsf{A}}, x_{\mathsf{B}})$.[8] The adversary then submits a value β, which is checked against the equation $\beta = \left(\sum_{i\in[n_{\mathsf{O}}]} r_i\alpha_i\right) \oplus \left(\sum_{i\in[n_{\mathsf{O}}]} r_i'\alpha_i'\right)$. This way, the adversary learns only a single bit of information (*i.e.*, the truth value of the equation) during verification.

Since r_i and r_i' are uniformly random, if $\beta = \left(\sum_{i\in[n_{\mathsf{O}}]} r_i\alpha_i\right) \oplus \left(\sum_{i\in[n_{\mathsf{O}}]} r_i'\alpha_i'\right)$ holds and the adversary knows this fact, this adversary must have known each α_i and α_i', except with negligible probability. Therefore, sending $\{(\alpha_i, \alpha_i')\}_{i\in[n_{\mathsf{O}}]}$ to the adversary after successful verification does not provide any additional information beyond what the adversary has already inferred.

Remark 1. The progressive revelation phase of $\mathcal{F}_{\mathsf{OneBitAdv}}$ can be straightforwardly modified to reveal k bits instead of a single bit in each iteration. Consequently, the resulting ideal functionality would limit the adversary's advantage to k bits. Protocols securely realizing this ideal functionality are actively secure with k-bit advantage bound. We also note that the variable k could even differ in each iteration.

4 Our One-Bit Advantage Bound Protocol

In this section, we first introduce the building blocks of our protocol. Then, we describe our protocol in detail.

[8] Note that notation g_i (resp. α_i) is essentially equivalent to g_i' (resp. α_i'); the use of *prime symbols* here serves solely to simplify description in our security proof.

4.1 Building Blocks

We first introduce the syntax of projective garbling schemes [4] as follows.

Definition 1. *A projective garbling scheme consists of five algorithms:*

- $(\mathsf{GC}, e, \mathsf{dk}) \leftarrow \mathsf{Gb}(1^\kappa, \mathcal{C})$. *The garbling algorithm* Gb *takes as input the security parameter* 1^κ *and a circuit* $\mathcal{C}$ *with* $n = n_\mathsf{A} + n_\mathsf{B}$ *input wires and* n_O *output wires. It outputs a garbled circuit* GC*, a mapping of input-wire labels* $e := \{(X_{i,0}, X_{i,1})\}_{i\in[n]}$*, and a decoding key* dk*.*
- $(d, D) := \mathsf{Dc}(\mathsf{dk})$. *The decoding information derivation algorithm* Dc *takes as input a decoding key* dk *and outputs the* simple *decoding information* d *and* authenticated *decoding information* D*.*
- $X := \mathsf{En}(e, x)$. *The encoding algorithm* En *takes as input the input-wire label mapping* $e = \{(X_{i,0}, X_{i,1})\}_{i\in[n]}$ *and input* $x \in \{0,1\}^n$*, and outputs input-wire labels* $X := \{X_{i,x_i}\}_{i\in[n]}$*.*
- $Y := \mathsf{Ev}(\mathsf{GC}, X)$. *The evaluation algorithm* Ev *takes as input the garbled circuit* GC *and the input-wire labels* $X = \{X_i\}_{i\in[n]}$*, and outputs the output-wire labels* $Y := \{Y_i\}_{i\in[n_\mathsf{O}]}$*.*
- $y := \mathsf{De}(d, Y)$. *The simple decoding algorithm* De *takes as input the simple decoding information* d *and output-wire labels* $Y = \{Y_i\}_{i\in[n_\mathsf{O}]}$*, and outputs the decoded result* y*.*
- $y := \mathsf{AuDe}(D, Y)$. *The authenticated decoding algorithm* AuDe *takes as input the authenticated decoding information* D *and output-wire labels* $Y = \{Y_i\}_{i\in[n_\mathsf{O}]}$*, and outputs the decoded result* y*. If decoding fails,* y *may take the value* $\perp$*.*

The scheme is correct if for any (polynomial-size) circuit $\mathcal{C}$ *and input* x*, after computing* $(\mathsf{GC}, e, \mathsf{dk}) \leftarrow \mathsf{Gb}(1^\kappa, \mathcal{C})$ *and* $(d, D) := \mathsf{Dc}(\mathsf{dk})$*, we have*

$$\mathcal{C}(x) = \mathsf{De}(d, \mathsf{Ev}(\mathsf{GC}, \mathsf{En}(e, x)))$$

and

$$\mathcal{C}(x) = \mathsf{AuDe}(D, \mathsf{Ev}(\mathsf{GC}, \mathsf{En}(e, x)))$$

except with negligible probability.[9]

Different from the garbling scheme definition in [4], we separate the decoding into simple decoding and authenticated decoding algorithms. This distinction aligns naturally with most existing garbling schemes. When context permits, we may combine the Gb and Dc algorithms into a single function: $(\mathsf{GC}, e, \mathsf{dk}, d, D) \leftarrow \mathsf{Gb}(1^\kappa, \mathcal{C})$.

For our protocol, we require that the garbling scheme adheres to the *point-permute-freeXOR-compatible* property, defined as follows:

Definition 2. *A projective garbling scheme along with an algorithm* $\mathsf{EvalAND}$ *for the security parameter* κ *is* point-permute-freeXOR-compatible *if:*

Point-and-Permute Compatibility. *Each wire* w *in the garbled circuitis associated with a point-permute bit* λ_w*. During garbled circuit evaluation, the wire label* $\mathsf{L}_{w,\hat{z}_w} \in \{0,1\}^\kappa$ *is derived, and the true bit for wire* w *is* $z_w = \lambda_w \oplus \hat{z}_w$*, where* $\hat{z}_w = \mathsf{lsb}(\mathsf{L}_{w,\hat{z}_w})$ *represents the (public) masked bit on* $\mathsf{L}_{w,\hat{z}_w}$*.*

FreeXOR Compatibility. *There is a global offset* $\Delta \in \{0,1\}^{\kappa-1}||1$*, such that the XOR of each wire label pair are* Δ *and the masked bit of* $\mathsf{L}_{w,\hat{z}_w}$ *is* $\hat{z}_w = \mathsf{lsb}(\mathsf{L}_{w,\hat{z}_w})$*. For any gate* $\mathcal{G} = (\alpha, \beta, \gamma, T)$*, where the evaluator holds* $\mathsf{L}_{\alpha,\hat{z}_\alpha}$ *and* $\mathsf{L}_{\beta,\hat{z}_\beta}$ *(alongside* $\hat{z}_\alpha$ *and* $\hat{z}_\beta$*), evaluation proceeds as follows:*
  - *If* $T = \oplus$*, compute* $\mathsf{L}_{\gamma,\hat{z}_\gamma} := \mathsf{L}_{\alpha,\hat{z}_\alpha} \oplus \mathsf{L}_{\beta,\hat{z}_\beta}$ *(with* $\hat{z}_\gamma := \hat{z}_\alpha \oplus \hat{z}_\beta$ *implicitly). The point-permute bit for wire* γ *is then* $\lambda_\gamma = \lambda_\alpha \oplus \lambda_\beta$*.*
  - *If* $T = \wedge$*, compute* $(\mathsf{L}_{\gamma,\hat{z}_\gamma}) := \mathsf{EvalAND}(\mathsf{L}_{\alpha,\hat{z}_\alpha}, \mathsf{L}_{\beta,\hat{z}_\beta})$*, where* $\mathsf{EvalAND}$ *is the multiplication gate evaluation algorithm that inputs the labels of the input wires for an AND gate, evaluates the garbled gate, and outputs the corresponding output-wire label.*

We denote $\mathsf{Gb}(1^\kappa, \mathcal{C}; \Delta)$ to specify the global offset Δ explicitly.

For a point-permute-freeXOR-compatible garbling scheme, such as the half-gates scheme [38] and three-halves scheme [32], the global offset Δ and a wire label L_{w,λ_w} (where $\lambda_w = \mathsf{lsb}(\mathsf{L}_{w,\lambda_w})$) suffice to determine output-wire labels as $Y_{i,b} := \mathsf{L}_{w,\lambda_w} \oplus b\Delta$ for $b \in \{0,1\}$, where $Y_{i,b}$ represents the label for the real bit b on the ith output wire. Consequently, we set the decoding key as $\mathsf{dk} = (\Delta, \{\mathsf{L}_{w,\lambda_w}\}_{w\in\mathcal{W}_\mathsf{O}})$. In our protocol, we define the simple decoding information as $d = \{\lambda_{\mathsf{out}(i)}\}_{i\in[n_\mathsf{O}]}$. Given output-wire labels $\{Y_i\}_{i\in[n_\mathsf{O}]}$, the output of $\mathsf{De}(d, Y)$ is $y_i := \mathsf{lsb}(Y_i) \oplus \lambda_w$, for $w = \mathsf{out}(i)$ and $i \in [n_\mathsf{O}]$.

We give the security definition for such garbling schemes as follows.

[9] We allow a negligible failure probability as the definition in [32].

Definition 3. *A projective point-permute-freeXOR-compatible garbling scheme achieves* obliviousness *if there is a probabilistic polynomial-time (*PPT*) simulator $\mathcal{S}_{\mathsf{Gb}}$, such that for any circuit $\mathcal{C}$ and input x, the outputs of the following two games are indistinguishable.*

$(\mathsf{GC}, e, \mathsf{dk}, d, D) \leftarrow \mathsf{Gb}(1^\kappa, \mathcal{C})$ $X := \mathsf{En}(e, x)$ return (GC, X)	$(\mathsf{GC}, X) \leftarrow \mathcal{S}_{\mathsf{Gb}}(1^\kappa, \mathcal{C})$ return (GC, X)

We adapt the privacy property introduced in [4] and define the *equivocable privacy* property as follows.

Definition 4. *A projective point-permute-freeXOR-compatible garbling scheme achieves* equivocable privacy *if there is a tuple of* PPT *simulators $(\mathcal{S}_{\mathsf{Gb}}, \mathcal{S}'_{\mathsf{Gb}})$, such that for any circuit $\mathcal{C}$ and input x, the output of the following games are indistinguishable.*

$(\mathsf{GC}, e, \mathsf{dk}, d, D) \leftarrow \mathsf{Gb}(1^\kappa, \mathcal{C})$ $X := \mathsf{En}(e, x)$ return (GC, X, d)	$(\mathsf{GC}, X, z) \leftarrow \mathcal{S}_{\mathsf{Gb}}(1^\kappa, \mathcal{C})$ $d \leftarrow \mathcal{S}'_{\mathsf{Gb}}(\mathcal{C}(x), z)$ return (GC, X, d)

Now, the simulator is split into two components $\mathcal{S}_{\mathsf{Gb}}$ and $\mathcal{S}'_{\mathsf{Gb}}$. The first component $\mathcal{S}_{\mathsf{Gb}}$ does not require the evaluation result $\mathcal{C}(x)$ as input and generates (GC, X) independently. Additionally, $\mathcal{S}_{\mathsf{Gb}}$ produces auxiliary data z, which could be $Y := \mathsf{Ev}(\mathsf{GC}, X)$ for most existing garbling schemes and is then used as input to $\mathcal{S}'_{\mathsf{Gb}}$. The second component $\mathcal{S}_{\mathsf{Gb}}$ takes $\mathcal{C}(x)$ and z as inputs to produce the simulated simple decoding information d. In other words, the simulated garbled circuit GC and garbled input X are generated initially, while the simulated simple decoding information d can be generated subsequently once the evaluation result $\mathcal{C}(x)$ is known, ensuring consistency with (GC, X). Notably, both the half-gates and three-halves garbling schemes satisfy equivocable privacy, as the simulator in their security proofs of privacy generates d with respect to $\mathcal{C}(x)$ after GC and X are generated. In our protocol, parties only receive d, and thus, we do not need to consider the authenticated decoding information D.

We introduce a relaxed form of authenticity property [4] for our protocol, termed *authenticity against one-bit leakage*.

Definition 5. *A projective point-permute-freeXOR-compatible garbling scheme for the security parameter κ achieves* authenticity against one-bit leakage *if for any circuit $\mathcal{C}$ and input x, no tuple of* PPT *adversaries ($\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$) can make the following game output* true *with non-negligible probability:*

$(\mathsf{GC}, e, \mathsf{dk}, d, D) \leftarrow \mathsf{Gb}(1^\kappa, \mathcal{C})$
$X := \mathsf{En}(e, x)$
$(\ell, \{g_i(\cdot)\}_{i\in[\ell]}, z_1) \leftarrow \mathcal{A}_1(\mathsf{GC}, X, d)$
$\alpha_i := g_i(\mathsf{dk})$, $r_i \leftarrow_\$ \mathbb{F}_2^\kappa$ for $i \in [\ell]$
$(\beta, z_2) \leftarrow \mathcal{A}_2(\{r_i\}_{i\in[\ell]}, z)$
If $\beta \neq \sum_{i\in[\ell]} r_i\alpha_i$, return false
Else $Y \leftarrow \mathcal{A}_3(\{\alpha_i\}_{i\in[\ell]}, z_2)$
return $\mathsf{AuDe}(D, Y) \notin \{\perp, \mathcal{C}(x)\}$

Unlike traditional authenticity, this relaxed definition allows an adversary to learn at most a single bit of information about dk. Using the idea of $\mathcal{F}_{\mathsf{OneBitAdv}}$, we allow the adversary to define a set of functions g_i applied to dk, specify a value β, and then learn if a random linear combination of the evaluations result of g_i, denoted by α_i, equals β. As a result, the adversary may learn at most one bit of information about dk. Since the linear combination is random, if the adversary confirms that the equation holds, it should know each α_i except with negligible probability. Therefore, we can safely provide the adversary $\mathcal{A}_3$ with α_i, and no additional information is leaked.

This relaxed definition is reasonable as it offers security guarantees similar to traditional authenticity. If an adversary could breach the authenticity against one-bit leakage with non-negligible probability, it can also randomly guess the leaked bit by itself, and thus with probability 50% the adversary might correctly guess this bit, enabling it to produce Y such that $\mathsf{AuDe}(D, Y) \notin \{\perp, \mathcal{C}(x)\}$ with non-negligible probability. We can easily verify that both the half-gates and three-halves garbling schemes satisfy authenticity against one-bit leakage.

Definition 6. *A projective point-permute-freeXOR-compatible garbling scheme is* secure *if it achieves obliviousness, equivocable privacy, and authenticity against one-bit leakage.*

Our protocol is built upon the components $\mathcal{F}_{\mathsf{Com}}$ for commitment, $\mathcal{F}_{\mathsf{OT}}$ for oblivious transfer, $\mathcal{F}_{\mathsf{OLE}}$ for oblivious linear evaluation, $\mathcal{F}_{\mathsf{Eq}}$ for equality testing, and $\mathcal{F}_{\mathsf{Rand}}$ for generating random values.

We use standard $\mathcal{F}_{\mathsf{Com}}$ and $\mathcal{F}_{\mathsf{OT}}$ within our protocol. For $\mathcal{F}_{\mathsf{OLE}}$, a sender inputs $\Delta \in \mathbb{F}_2^\kappa$ and $v \in \mathbb{F}_2^\kappa$, while a receiver inputs $u \in \mathbb{F}_2^\kappa$ and learns $w := v \oplus u\Delta$. For $\mathcal{F}_{\mathsf{Eq}}$, two parties input $x \in \{0,1\}^\kappa$ and $y \in \{0,1\}^\kappa$ respectively and learn whether $x = y$. For $\mathcal{F}_{\mathsf{Rand}}$ with parameter ℓ, the functionality outputs random $r_i \leftarrow\!\!\$\, \mathbb{F}_2^\kappa$ for $i \in [\ell]$ to both parties.

4.2 Our Scheme

We give a high-level description of our protocol $\Pi_{\mathsf{OneBitAdv}}$ below, and the formal description of our protocol is given in Figure 1.

The protocol $\Pi_{\mathsf{OneBitAdv}}$ employs a secure projective garbling scheme that is point-permute-freeXOR-compatible. It consists of three main phases: evaluation, verification, and output progressive revelation.

- **Evaluation:** Each party begins by independently generating a garbled circuit for the agreed-upon circuit $\mathcal{C}$ using their respective global keys Δ_{A} and Δ_{B}. They then execute two instances of garbled circuits, swapping their roles between the two instances. After evaluating the garbled circuit generated by the other party, each party derives the garbled output. They then compute the respective SPDZ-like MACs A_i and B_i based on both the derived garbled outputs and the output-wire labels of their own garbled circuits, as outlined in Section 2.3.
- **Verification:** Both parties use $\mathcal{F}_{\mathsf{OLE}}$ to jointly generate an authenticated secret sharing of a random value, which serves to mask the linear combination, as detailed in Section 2.3. Additionally, they obtain a set of random values over $\mathbb{F}_2^\kappa$ from $\mathcal{F}_{\mathsf{Rand}}$. Using this set of values as coefficients, they then perform a random linear combination on their shares and MACs of the random value and the output bits of both circuits, where the same MAC applied to corresponding output bits from both circuits. The result of this linear combination is then revealed, and $\mathcal{F}_{\mathsf{Eq}}$ is applied to confirm the validity of this opening. If the equality test fails, the protocol halts immediately, signaling potential cheating.
- **Output Progressive Revelation:** In this final phase, both parties jointly reveal the evaluation result bit by bit. For each output bit, the parties exchange the point-permute bit associated with that wire and reconstruct the output bit by combining the received point-permute bit with the masked bit derived from evaluating the other party's garbled circuit. An equality test is then conducted to validate this opening. The protocol aborts if any opening is invalid. This procedure thus ensures that the adversary's advantage is strictly limited to a single bit.

Theorem 1. *The protocol $\Pi_{\mathsf{OneBitAdv}}$ along with a secure projective garbling scheme that is point-permute-freeXOR-compatible securely realizes functionality $\mathcal{F}_{\mathsf{OneBitAdv}}$ against* PPT *malicious (rushing) adversaries in the ($\mathcal{F}_{\mathsf{Com}}$, $\mathcal{F}_{\mathsf{OT}}$, $\mathcal{F}_{\mathsf{OLE}}$, $\mathcal{F}_{\mathsf{Rand}}$, $\mathcal{F}_{\mathsf{Eq}}$)-hybrid world.*

We give the sketch proof of this theorem as follows.

Proof. Without loss of generality, we assume that P_{B} is honest, while P_{A} is corrupted by the adversary $\mathcal{A}$. The roles of P_{A} and P_{B} in protocol $\Pi_{\mathsf{OneBitAdv}}$ are symmetric, so a similar proof applies when P_{B} is corrupted.

For an adversary $\mathcal{A}$ corrupting P_{A} in the ($\mathcal{F}_{\mathsf{Com}}$, $\mathcal{F}_{\mathsf{OT}}$, $\mathcal{F}_{\mathsf{OLE}}$, $\mathcal{F}_{\mathsf{Rand}}$, $\mathcal{F}_{\mathsf{Eq}}$)-hybrid world, we construct a simulator $\mathcal{S}$, which plays the role of P_{B} and runs $\mathcal{A}$ as a subroutine with auxiliary input z, interacting with $\mathcal{F}_{\mathsf{OneBitAdv}}$ in the ideal world. The simulation is given Figure 2. We now need to prove that the joint distribution of the view of $\mathcal{A}$ and the output of P_{B} in the ideal world is computationally indistinguishable from the joint distribution of the view of $\mathcal{A}$ and the output of P_{B} in the real protocol execution. We give the details in Appendix A.

Remark 2. To achieve a k-bit advantage bound, both parties can reveal k bits of the evaluation result in each iteration and verify their correctness. This verification can be conducted in batches using random linear combinations, as discussed in Section 2.3.

Protocol $\Pi_{\mathsf{OneBitAdv}}$

Public inputs: Both parties agree on the security parameter κ and a circuit $\mathcal{C}$, which has $n = n_{\mathsf{A}} + n_{\mathsf{B}}$ input wires and n_{O} output wires.
Private inputs: P_{A} has input $x_{\mathsf{A}} \in \{0,1\}^{n_{\mathsf{A}}}$. P_{B} has input $x_{\mathsf{B}} \in \{0,1\}^{n_{\mathsf{B}}}$.

Evaluation

1 P_{A} picks $\Delta_{\mathsf{A}} \leftarrow\!\!\$\ \mathbb{F}_2^\kappa$, such that $\mathsf{lsb}(\Delta_{\mathsf{A}}) = 1$. P_{B} also samples $\Delta_{\mathsf{B}} \leftarrow\!\!\$\ \mathbb{F}_2^\kappa$ in the form that $\mathsf{lsb}(\Delta_{\mathsf{B}}) = 1$.
2 P_{A} generates the garbled circuit GC via $(\mathsf{GC}, e, \mathsf{dk}, d, D) \leftarrow \mathsf{Gb}(1^\kappa, \mathcal{C}; \Delta_{\mathsf{A}})$, where $e = \{(X_{i,0}, X_{i,1})\}_{i\in[n]}$. Similarly, P_{B} generates the garbled circuit GC' via $(\mathsf{GC}', e', \mathsf{dk}', d', D') \leftarrow \mathsf{Gb}(1^\kappa, \mathcal{C}; \Delta_{\mathsf{B}})$, where $e' = \{(X'_{i,0}, X'_{i,1})\}_{i\in[n]}$. Here dk and dk' can be used to derive point-permute bits (resp. output-wire labels) λ_w and λ'_w (resp. $\mathsf{L}_{w,0}$ and $\mathsf{L}'_{w,0}$) for each wire $w \in \mathcal{W}_{\mathsf{O}}$, respectively.
Both P_{A} and P_{B} then commit to their garbled circuits via $\mathcal{F}_{\mathsf{Com}}$.
3 P_{B}, as the sender, sends $\{(X'_{i,0}, X'_{i,1})\}_{i\in[n_{\mathsf{A}}]}$ to $\mathcal{F}_{\mathsf{OT}}$, while P_{A}, acting as the receiver, sends x_{A} and receives $\{X'_i := X'_{i,x_{\mathsf{A}}[i]}\}_{i\in[n_{\mathsf{A}}]}$. Meanwhile, two parties switch their roles, with P_{A} sending $\{(X_{i,0}, X_{i,1})\}_{i\in[n_{\mathsf{A}}+1,n]}$ for P_{B}'s input-wire labels, and P_{B}, as the OT receiver, sending x_{B} and receiving $\{X_i := X_{i,x_{\mathsf{B}}[i-n_{\mathsf{A}}]}\}_{i\in[n_{\mathsf{A}}+1,n]}$.
4 Each party sends the label they generated corresponding to their own input to the other party, *i.e.*, P_{A} sends $\{X_i := X_{i,x_{\mathsf{A}}[i]}\}_{i\in[n_{\mathsf{A}}]}$ to P_{B}, and P_{B} sends $\{X'_i := X'_{i,x_{\mathsf{B}}[i-n_{\mathsf{A}}]}\}_{i\in[n_{\mathsf{A}}+1,n]}$ to P_{A}.
5 Both parties open their garbled circuits GC and GC' to each other via $\mathcal{F}_{\mathsf{Com}}$. P_{A} evaluates GC' by computing $Y' := \mathsf{Ev}(\mathsf{GC}', X')$, where $X' = \{X'_i\}_{i\in[n]}$ consists of all input-wire labels P_{A} has retrieved from $\mathcal{F}_{\mathsf{OT}}$ and received from P_{B}. For each $w = \mathsf{out}(i)$, where $i \in [n_{\mathsf{O}}]$, P_{A} sets $a_{i,2} := \hat{z}'_w := \mathsf{lsb}(Y'_i)$ and $A_{i,2} := Y'_i$. Simultaneously, P_{B} defines $b_{i,2} := \lambda'_w$ and $B_{i,2} := \mathsf{L}'_{w,0} \oplus \lambda'_w \Delta_{\mathsf{B}}$.
Meanwhile, P_{B} evaluates GC by computing $Y := \mathsf{Ev}(\mathsf{GC}, X)$, where X contains the input-wire labels P_{B} has retrieved and received. For each $i \in [n_{\mathsf{O}}]$, P_{B} sets $b_{i,1} := \hat{z}_w := \mathsf{lsb}(Y_i)$ and $B_{i,1} := Y_i$, while P_{A} defines $a_{i,1} := \lambda_w$ and $A_{i,1} := \mathsf{L}_{w,0} \oplus \lambda_w \Delta_{\mathsf{A}}$.
P_{A} and P_{B} compute $A_i := A_{i,1} \oplus A_{i,2}$ and $B_i := B_{i,1} \oplus B_{i,2}$ for $i \in [n_{\mathsf{O}}]$, respectively.

Verification

6 P_{A} randomly selects $\mathsf{L}_0 \leftarrow\!\!\$\ \mathbb{F}_2^\kappa$, and, as the sender, sends L_0 and Δ_{A} to $\mathcal{F}_{\mathsf{OLE}}$. Meanwhile, P_{B} sends a random $b_0 \leftarrow\!\!\$\ \mathbb{F}_2^\kappa$ to $\mathcal{F}_{\mathsf{OLE}}$ and receives $B_{0,1} = \mathsf{L}_0 \oplus b_0 \Delta_{\mathsf{A}}$. The parties switch roles, with P_{B} sending $\mathsf{L}'_0 \leftarrow\!\!\$\ \mathbb{F}_2^\kappa$ and Δ_{B} to $\mathcal{F}_{\mathsf{OLE}}$, while P_{A} sends $a_0 \leftarrow\!\!\$\ \mathbb{F}_2^\kappa$ and receives $A_{0,2} = \mathsf{L}'_0 \oplus a_0 \Delta_{\mathsf{B}}$ as the receiver.
P_{B} computes $B_0 := B_{0,1} \oplus \mathsf{L}'_0 \oplus b_0 \Delta_{\mathsf{B}}$, while P_{A} computes $A_0 := A_{0,2} \oplus \mathsf{L}_0 \oplus a_0 \Delta_{\mathsf{A}}$.
7 Both parties call $\mathcal{F}_{\mathsf{Rand}}$ to obtain random values $r_i \leftarrow\!\!\$\ \mathbb{F}_2^\kappa$ and $r'_i \leftarrow\!\!\$\ \mathbb{F}_2^\kappa$ for $i \in [n_{\mathsf{O}}]$.
P_{A} then computes $A := A_0 \oplus \left(\sum_{i\in[n_{\mathsf{O}}]} r_i A_i\right) \oplus \left(\sum_{i\in[n_{\mathsf{O}}]} r'_i A_i\right)$ and $a := a_0 \oplus \left(\sum_{i\in[n_{\mathsf{O}}]} r_i a_{i,1}\right) \oplus \left(\sum_{i\in[n_{\mathsf{O}}]} r'_i a_{i,2}\right)$, and sends a to P_{B}. P_{B} also computes $B := B_0 \oplus \left(\sum_{i\in[n_{\mathsf{O}}]} r_i B_i\right) \oplus \left(\sum_{i\in[n_{\mathsf{O}}]} r'_i B_i\right)$ and $b := b_0 \oplus \left(\sum_{i\in[n_{\mathsf{O}}]} r_i b_{i,1}\right) \oplus \left(\sum_{i\in[n_{\mathsf{O}}]} r'_i b_{i,2}\right)$, and sends b to P_{A}.
8 P_{A} and P_{B} input $A \oplus (a \oplus b)\Delta_{\mathsf{A}}$ and $B \oplus (a \oplus b)\Delta_{\mathsf{B}}$ to $\mathcal{F}_{\mathsf{Eq}}$, respectively, to verify if the values match. If $\mathcal{F}_{\mathsf{Eq}}$ returns false, parties output $\mathsf{cheating}$ and the protocol aborts with termination.

Output Progressive Revelation

9 For $i \in [n_{\mathsf{O}}]$, P_{A} and P_{B} follow the procedure below to derive each bit y_i of the evaluation result y.
   (a) P_{A} sends $a_{i,1} = \lambda_w$ to P_{B}, while P_{B} sends $b_{i,2} = \lambda'_w$ to P_{A} for $w = \mathsf{out}(i)$. P_{A} computes $y'_i := a_{i,2} \oplus b_{i,2}$ and P_{B} computes $y_i := a_{i,1} \oplus b_{i,1}$.
   (b) P_{A} inputs $A_i \oplus y'_i \Delta_{\mathsf{A}}$ to $\mathcal{F}_{\mathsf{Eq}}$, and P_{B} inputs $B_i \oplus y_i \Delta_{\mathsf{B}}$ to $\mathcal{F}_{\mathsf{Eq}}$. If $\mathcal{F}_{\mathsf{Eq}}$ returns false, parties output $\perp$ with abortion. Otherwise, the ith bit of the evaluation result is output by both P_{A} and P_{B} as y'_i and y_i, respectively.

Fig. 1: Our constant-round 2PC protocol achieving active security with one-bit advantage bound in the ($\mathcal{F}_{\mathsf{Com}}$, $\mathcal{F}_{\mathsf{OT}}$, $\mathcal{F}_{\mathsf{OLE}}$, $\mathcal{F}_{\mathsf{Rand}}$, $\mathcal{F}_{\mathsf{Eq}}$)-hybrid world.

1 $\mathcal{S}$ samples $\Delta_{\mathsf{B}} \leftarrow\!\!\$\ \mathbb{F}_2^\kappa$, such that $\mathsf{lsb}(\Delta_{\mathsf{B}}) = 1$, as in the protocol.

2 $\mathcal{S}$ computes $(\mathsf{GC}', X', z) \leftarrow \mathcal{S}_{\mathsf{Gb}}(1^\kappa, \mathcal{C})$, where $X' = \{X'_i\}_{i\in[n_{\mathsf{O}}]}$. Let $Y' = \{Y'_i\}_{i\in[n_{\mathsf{O}}]} = \mathsf{Ev}(\mathsf{GC}', X')$. For each output wire $w = \mathsf{out}(i)$, $\mathcal{S}$ defines $\hat{z}'_w = \mathsf{lsb}(Y'_i)$.
Acting as $\mathcal{F}_{\mathsf{Com}}$, $\mathcal{S}$ informs the adversary $\mathcal{A}$ that the honest party P_{B} has committed his garbled circuit to $\mathcal{F}_{\mathsf{Com}}$. Subsequently, $\mathcal{S}$ obtains the garbled circuit GC from $\mathcal{A}$.

3 If P_{A} is the receiver in $\mathcal{F}_{\mathsf{OT}}$, $\mathcal{S}$ receives x_{A} from $\mathcal{A}$, sends $\{X'_i\}_{i\in[n_{\mathsf{A}}]}$ to $\mathcal{A}$. Then $\mathcal{S}$ sends x_{A} to $\mathcal{F}_{\mathsf{OneBitAdv}}$. If P_{A} acts as the sender in $\mathcal{F}_{\mathsf{OT}}$, $\mathcal{S}$ receives $\{(X_{i,0}, X_{i,1})\}_{i\in[n_{\mathsf{A}}+1,n]}$ from $\mathcal{A}$.

4 $\mathcal{S}$ sends $\{X'_i\}_{i\in[n_{\mathsf{A}}+1,n]}$ to $\mathcal{A}$ and receives $\{X_i\}_{i\in[n_{\mathsf{A}}]}$ from $\mathcal{A}$.

5 Acting as $\mathcal{F}_{\mathsf{Com}}$, $\mathcal{S}$ opens GC' to $\mathcal{A}$.

6 In the role of receiver in $\mathcal{F}_{\mathsf{OLE}}$, $\mathcal{S}$ receives L_0 and $\hat{\Delta}_{\mathsf{A}}$ from $\mathcal{A}$. As the sender, $\mathcal{S}$ receives a_0 from $\mathcal{A}$ and returns $A_{0,2} \leftarrow\!\!\$\ \mathbb{F}_2^\kappa$ to $\mathcal{A}$. Set $\mathsf{L}'_0 := A_{0,2} \oplus a_0\Delta_{\mathsf{B}}$.

7 For each $i \in [n_{\mathsf{O}}]$, $\mathcal{S}$ defines the Boolean function g_i (resp. g'_i) with fixed input x_{A} and variable $x_{\mathsf{B}} \in \{0,1\}^{n_{\mathsf{B}}}$ as follows. Note that corresponding values are hard-coded in g_i and g'_i.
   (a) Let $X_i := X_{i,x_{\mathsf{B}}[i-n_{\mathsf{A}}]}$ for $i \in [n_{\mathsf{A}}+1, n]$.
   (b) Compute $Y := \mathsf{Ev}(\mathsf{GC}, X)$, where $X = \{X_i\}_{i\in[n]}$.
   (c) Define $B_{i,1} := Y_i$ and $b_{i,1} := \hat{z}_w = \mathsf{lsb}(Y_i)$ for $w = \mathsf{out}(i)$.
   (d) Compute $y := \mathcal{C}(x_{\mathsf{A}}, x_{\mathsf{B}})$, where $y = (y_1, \cdots, y_{n_{\mathsf{O}}}) \in \{0,1\}^{n_{\mathsf{O}}}$.
   (e) Let $\lambda'_w := y_i \oplus \hat{z}'_w$ (*i.e.*, this is what is done by $\mathcal{S}'_{\mathsf{Gb}}$ with respect to equivocable privacy), and define $b_{i,2} := \lambda'_w$ for $w = \mathsf{out}(i)$.
   (f) Set $B_{i,2} := Y'_i \oplus y_i\Delta_{\mathsf{B}}$ and $B_i := B_{i,1} \oplus B_{i,2}$.
   (g) Output $\alpha_i := B_i \oplus b_{i,1}\hat{\Delta}_{\mathsf{A}} \oplus b_{i,1}\Delta_{\mathsf{B}}$ (resp. $\alpha'_i := B_i \oplus b_{i,2}\hat{\Delta}_{\mathsf{A}} \oplus b_{i,2}\Delta_{\mathsf{B}}$).
   $\mathcal{S}$ sends $\{g_i, g'_i\}_{i\in[n_{\mathsf{O}}]}$ to $\mathcal{F}_{\mathsf{OneBitAdv}}$, receives random values $\{r_i, r'_i\}_{i\in[n_{\mathsf{O}}]}$, and forwards these values to $\mathcal{A}$ with respect to $\mathcal{F}_{\mathsf{Rand}}$.
   $\mathcal{S}$ randomly selects $b \leftarrow\!\!\$\ \mathbb{F}_2^\kappa$ and sends it to $\mathcal{A}$, while receiving $a \in \mathbb{F}_2^\kappa$ from $\mathcal{A}$.

8 $\mathcal{S}$ receives $\hat{\beta} \in \mathbb{F}_2^\kappa$ from $\mathcal{A}$ with respect to $\mathcal{F}_{\mathsf{Eq}}$, computes $\beta := \hat{\beta} \oplus b\hat{\Delta}_{\mathsf{A}} \oplus \mathsf{L}_0 \oplus \mathsf{L}'_0 \oplus a\Delta_{\mathsf{B}}$, sends it to $\mathcal{F}_{\mathsf{OneBitAdv}}$, and receives the output. If the output is false, $\mathcal{S}$ simulates rejection by honest P_{B}. Otherwise, receive $\{(\alpha_i, \alpha'_i)\}_{i\in[n_{\mathsf{O}}]}$ from $\mathcal{F}_{\mathsf{OneBitAdv}}$.

9 For each $i \in [n_{\mathsf{O}}]$, $\mathcal{S}$ follows the procedure below.
   (a) $\mathcal{S}$ receives y_i from $\mathcal{F}_{\mathsf{OneBitAdv}}$, computes $b_{i,2} = \lambda'_w := y_i \oplus \hat{z}'_w$ for $w = \mathsf{out}(i)$, sends it to $\mathcal{A}$, and receives $a_{i,1}$ from $\mathcal{A}$.
   (b) $\mathcal{S}$ receives $\tilde{\alpha}_i$ with respect to $\mathcal{F}_{\mathsf{Eq}}$. Then $\mathcal{S}$ computes $\hat{b}_{i,1} := y_i \oplus a_{i,1}$ and checks if $\alpha_i \oplus \hat{b}_{i,1}\Delta_{\mathsf{B}} \oplus \hat{b}_{i,1}\hat{\Delta}_{\mathsf{A}} = \alpha'_i \oplus b_{i,2}\Delta_{\mathsf{B}} \oplus b_{i,2}\hat{\Delta}_{\mathsf{A}}$. If the equation does not hold, $\mathcal{S}$ simulates rejection by P_{B} and sends abort to $\mathcal{F}_{\mathsf{OneBitAdv}}$.
   Then $\mathcal{S}$ checks if $\tilde{\alpha}_i = \alpha_i \oplus a_{i,1}\Delta_{\mathsf{B}} \oplus a_{i,1}\hat{\Delta}_{\mathsf{A}} \oplus y_i\hat{\Delta}_{\mathsf{A}}$. If the equation does not hold, $\mathcal{S}$ simulates rejection by P_{B} and sends abort to $\mathcal{F}_{\mathsf{OneBitAdv}}$. Otherwise, $\mathcal{S}$ sends $\mathsf{continue}$ to $\mathcal{F}_{\mathsf{OneBitAdv}}$, outputs whatever $\mathcal{A}$ outputs and continues.

Fig. 2: The simulator $\mathcal{S}$ with respect to $\mathcal{F}_{\mathsf{OneBitAdv}}$ in the ideal world.

5 Improvements and Optimizations

Our protocol is actively secure and already highly efficient, but we briefly discuss some additional improvements and optimizations that can be further applied. As with other 2PC protocols utilizing garbled circuits, we can leverage cost-effective correlated oblivious transfer and oblivious linear evaluation to improve efficiency. We can also apply circuit pipelining, allowing parties to transmit garbled gates while simultaneously evaluating the garbled circuit. Additionally, we introduce approaches specifically applicable to our protocol as follows.

Leveraging $\mathcal{F}_{\mathsf{Rand}}$ for free. Since we use standard garbling schemes, garbled circuits GC and GC' in the protocol have high entropy. Therefore, two parties can generate pseudorandom values non-interactively by employing a pseudorandom function, where GC and GC' are collectively used to derive the key, to generate necessary random values in CTR mode.

Randomize order of output revelation. We can further limit adversaries' advantage by randomizing the order of output bit revelation. Both parties can collaboratively generate a random permutation P over $[n_{\mathsf{O}}]$, defining the sequence for output bit disclosure. We can also use a similar method to determine which party reveals the output bits first for each iteration. The required randomness can be generated non-interactively, as described above.

Optimizations in the random-oracle model. Since our protocol uses standard garbling schemes, all values involved in the equality test maintain high entropy when unknown to the adversary. Therefore, to perform an equality test with inputs a and b, both parties can simply commit to and open $\mathsf{H}(a)$ and $\mathsf{H}(b)$, respectively, where H is modeled as a random oracle, then verify if $\mathsf{H}(a) = \mathsf{H}(b)$. We can also use the random oracle for commitment by defining $\mathsf{Com}(m; r) = \mathsf{H}(m, r)$, where $r \leftarrow\!\!\$\, \{0,1\}^{\kappa}$.

With this setup, we can further reduce rounds in the progressive output revelation phase. For the ith output, P_{A} can first commit to both $\mathsf{H}(A_i)$ and $\mathsf{H}(A_i \oplus \Delta_{\mathsf{A}})$ using the same randomness r_i, while P_{B} commits to $\mathsf{H}(B_i)$ and $\mathsf{H}(B_i \oplus \Delta_{\mathsf{B}})$ using randomness r'_i. Note that it is essential to ensure that commitments from the two parties are distinct to prevent an adversary from copying the honest party's commitments and decommitments. They decommit by exchanging r_i and r'_i, allowing each party to derive the output bit: for instance, if $\mathsf{H}(A_i)$ matches P_{B}'s first commitment, P_{A} concludes $y_i = 0$; if $\mathsf{H}(A_i \oplus \Delta_{\mathsf{A}})$ is P_{B}'s second committed value, then $y_i = 1$. If neither matches, the output is rejected. Furthermore, commitments for the ith output bit can be sent alongside decommitments for the $(i-1)$-th output, thus requiring only $n_{\mathsf{O}} + 1$ rounds to reveal and verify all output bits. In protocols with k-bit advantage bound, enumerating all possible k-bit outputs may be relatively expensive. Therefore, both parties can simply perform k parallel iterations of 1-bit revelation.

6 Performance

6.1 Comparison

In the following, we argue that each phase of our protocol incurs low additional overhead compared to the passively secure counterparts.

Evaluation. Steps 1 to 4 are nearly identical to those in passively secure protocols, aside from the use of actively secure OT protocols. It is well-known that actively secure OTs are highly efficient and perform comparably to passively secure OTs in both local area network (LAN) and wide area network (WAN). If the input lengths of the two parties are asymmetric, *e.g.*, if P_{A} has a longer input while P_{B}'s input is shorter, then passively secure protocols may require fewer OTs. But we note that OTs are employed solely for retrieving input-wire labels, so for large circuits, they contribute only a small fraction of the overall protocol cost. Sending and evaluating garbled circuits in Step 5 remain the same as in passively secure protocols. The primary difference of Step 5 lies in computing $(A_{i,1}, A_i)$ and $(B_{i,2}, B_i)$; however, only XOR operations over $\{0,1\}^{\kappa}$ are required, amounting to $2n_{\mathsf{O}}$ XORs per party. This added overhead is minimal. It is easy to see that this phase requires only a small constant number of rounds.

Verification. The execution cost of two oblivious linear evaluations in Step 6 is constant. As discussed in Section 5, randomness in Step 7 can be generated efficiently and non-interactively. Given that $a_{i,1}$, $a_{i,2}$, $b_{i,1}$, and $b_{i,2}$ are bits, the computations of a and b in Step 7 involve only XOR

operations over $\{0,1\}^\kappa$, with $2n_{\mathsf{O}}$ XORs per party. For the computations of A and B, each party performs $2n_{\mathsf{O}}$ multiplications and additions (*i.e.*, XORs) over $\mathbb{F}_2^\kappa$. Therefore, the computational overhead of these operations is minimal. The communication cost of Step 7 involves only the exchange of two elements in $\mathbb{F}_2^\kappa$. For the equality test in Step 8, employing the optimizations described in Section 5, one hash value (serving as a commitment) and a single κ-bit opening are required; there is no need to send the committed hash value for the equality test since, if the test passes, the other party must possess this value. This phase also requires only a small constant number of rounds.

Output Progressive Revelation. Utilizing the optimizations via random oracles in Section 5, the output progressive revelation phase can be implemented in $n_{\mathsf{O}}+1$ rounds, which is *nearly optimal* for realizing output progressive revelation in 2PC. In each round, both parties compute two hash values, exchanging these along with one κ-bit opening. Both the communication and computation costs are minimal. The same efficiency also applies when releasing k bits of output per round.

Moreover, we note that upon successful completion of the verification phase, both parties are secretly sharing a consistent and correct evaluation result. We assert the correctness of the evaluation result here because garbled circuits generated by honest parties must yield correct output. As a result, even if all n_{O} bits of the evaluation result are revealed in a single iteration (*i.e.*, under n_{O}-bit advantage bound) during output progressive revelation and the adversary forces an abort, the adversary gains only the additional knowledge that its garbled circuit output aligns with that of the honest party. This differs from one-bit-leakage protocols, where adversary could learn the correct evaluation result even if the equality test fails. The security guarantees of our protocol approach those of traditional active security with abort. In addition, achieving verifiable output progressive revelation is a non-trivial task for actively secure protocols, and it often involves additional overhead.

Therefore, our protocol achieves both high efficiency and high security guarantees.

6.2 Implementation and Evaluation

We provide a proof-of-concept implementation to demonstrate the performance of our protocol. All experiments are conducted on an `ecs.hfr7.4xlarge` instance of Alibaba Cloud, equipped with 128 GiB of memory. Each party is run on a 3.69 GHz vCPU core in single-threaded mode. The protocol is evaluated under both LAN and WAN settings: in the LAN configuration, the network bandwidth is 2 Gbps with 0.1 ms latency; in the WAN configuration, the bandwidth is 200 Mbps with 60 ms latency. Our implementation builds upon the `EMP-Toolkit` [33].

This implementation includes methods for non-interactive randomness generation and optimizations in the random-oracle model, as described in Section 5. Specifically, we utilize `AES-NI` in CTR mode as a pseudorandom generator (PRG) and `SHA-256` to instantiate the random oracle. Given recent results [16,6] that the half-gates scheme [38] outperforms the three-halves scheme [32] in many practical scenarios and has more mature implementations, we use the half-gates scheme [38] in our proof-of-concept implementation to benchmark against passively secure protocols that also employ this scheme. We note that comparable results should hold for the three-halves scheme [32].

Table 1: Evaluated Boolean circuits. The parameters n_{A}, n_{B}, and n_{O} denote the bit lengths of P_{A}'s input, P_{B}'s input, and the circuit output, respectively. The total number of gates and AND gates are also listed.

Circuit	n_{A}	n_{B}	n_{O}	#Total gates	#AND gates
AES-128	128	128	128	33,616	6,800
SHA-128	256	256	160	106,601	37,300
SHA-256	256	256	256	236,112	90,825
Hamming Dist.	1,048,576	1,048,576	22	8,388,524	2,097,130
Integer Mult.	2,048	2,048	2,048	12,568,585	4,192,257
Sorting	131,072	131,072	131,072	56,903,681	10,223,616

We compare the running time and communication overhead of our protocol with state-of-the-art passively and actively secure 2PC implementations based on garbled circuits (GC). The running time

includes both computational costs and network I/O, while the communication overhead is defined as the cumulative size of all messages exchanged between the parties. To ensure a fair comparison with protocols lacking progressive revelation, we release all outputs simultaneously during the progressive revelation phase of our protocol. Our comparison covers circuits are commonly used in previous work (*e.g.*, [9,20,13]) and representative of the primary categories encountered in practical applications. Table 1 details these circuits, either sourced from `SCALE-MAMBA` [25] or generated using `EMP-Toolkit` [33]. Specifically, the Boolean circuit `AES-128` computes the encrypted value of a 128-bit input using a 128-bit key, while `SHA-128` and `SHA-256` compute the hash values of 128-bit and 256-bit inputs, respectively. The Hamming distance circuit calculates the Hamming distance between two 2^{20}-bit large integers. The integer multiplication circuit involves multiplying two 2048-bit large integers. The sorting circuit sorts an array containing 4096 32-bit integers using the bitonic sorting algorithm. The number of AND gates for `AES-128`, `SHA-128`, and `SHA-256` ranges from 10^3 to 10^5, which corresponds to small circuits, while the number of AND gates for Hamming distance, integer multiplication, and sorting ranges from 10^6 to 10^7, which corresponds to large circuits. Note that the input sizes for Hamming distance, integer multiplication, and sorting are custom-defined and widely used as benchmarks in MPC literature.

Table 2: Comparison of running time between the passively and actively secure GC-based 2PC protocol and our protocol in LAN.

Circuits	Runtime for LAN (ms)				
	Ours	Passive 2PC [38]	Slowdown	Active 2PC [34]	Speedup
AES-128	5.186	3.917	1.324×	35.765	6.896×
SHA-128	21.420	20.109	1.065×	170.425	7.956×
SHA-256	50.800	49.177	1.033×	408.673	8.045×
Hamming Dist.	1,130.237	1,125.191	1.004×	10,386.330	9.190×
Integer Mult.	2,134.655	2,119.995	1.092×	18,481.636	8.658×
Sorting	5,687.642	5,676.164	1.002×	46,234.423	8.129×

Table 3: Comparison of running time between the passively and actively secure GC-based 2PC protocol and our protocol in WAN.

Circuits	Runtime for WAN (ms)				
	Ours	Passive 2PC [38]	Slowdown	Active 2PC [34]	Speedup
AES-128	355.447	21.664	16.407×	636.300	1.790×
SHA-128	555.088	164.875	3.366×	1,291.858	2.328×
SHA-256	782.826	266.713	2.935×	2,648.022	3.383×
Hamming Dist.	5,742.006	3,645.433	1.575×	60,855.122	10.598×
Integer Mult.	11,460.162	7,353.140	1.559×	108,088.760	9.431×
Sorting	26,767.659	17,687.417	1.513×	237,419.610	8.870×

Running Time in the LAN and WAN Setting. Tables 2 and 3 compares the running time of the proposed 2PC protocol with passively and actively secure GC-based 2PC protocols in both LAN and WAN. Compared to the passively secure GC-based 2PC protocol [38], the results indicate that over a LAN, our protocol incurs an overhead of up to 32.4%, consistent with our prior analysis. In a WAN environment, however, the overhead more than doubles for small circuits (*i.e.*, `AES-128`, `SHA-128`, and `SHA-256`) and remains around 50% for larger circuits (*i.e.*, Hamming Distance, Integer Multiplication, and Sorting). This elevated overhead for smaller circuits is because network I/O dominates the total running time for small-size circuits. As shown in Figure 1, our protocol consists of three phases: 1) evaluation, 2) verification, and 3) output progressive revelation. For small circuits such as `AES-128`, the evaluation phase takes 37.21 ms, while the verification and output progressive revelation phases together require 318.237 ms, accounting for 89.53% of the total running time.The overhead of the

latter two phases is mainly due to the 5 rounds of communication introduced, with each round incurring a 60 ms delay in the WAN setting. On the other hand, for larger circuits, the evaluation phase of the protocol constitutes over 95% of the total running time. The other two phases have minimal running time, but due to the 200 Mbps bandwidth limit and cost of synchronization in the WAN setting, the evaluation phase of our protocol, which requires parallel instance computation, is more significantly impacted by the network limitation.

Compared to the state-of-the-art actively secure GC-based 2PC protocol implementation [34], our protocol achieves significantly better running time in both LAN and WAN settings, as expected, with an improvement factor of 6.9−10.6×, except for small circuits (*i.e.*, `AES-128`, `SHA-128`, and `SHA-256`) in the WAN setting, where the improvement is only 1.8−3.4×. The reason for this is similar to the above. For small circuits in the WAN setting, the verification and output progressive revelation phases introduced by our protocol add approximately 300 ms of network I/O time, which is on the same order of magnitude as the running time of the evaluation phase. This results in some communication advantage of our protocol being offset by the overhead introduced by these phases. However, in the LAN setting or for large circuit evaluations in the WAN setting, network I/O time accounts for less than 5%, which allows our protocol to achieve significant performance improvements due to the low-overhead circuit computation phase. This indicates that our protocol is highly efficient compared to existing actively secure GC-based 2PC protocols.

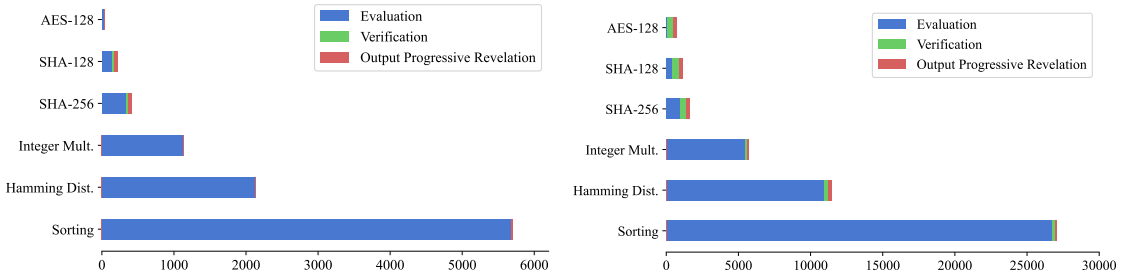
Table 4: Comparison of communication costs (MiB) between passively and actively secure GC-based 2PC protocols and our protocol, where $\mathsf{P_A}$ is the garbler and $\mathsf{P_B}$ is the evaluator in the passively and actively secure GC-based 2PC protocols.

Circuits	Passive 2PC [38]			Our Protocol			Active 2PC [34]		
	$\mathsf{P_A}$'s	$\mathsf{P_B}$'s	Total	$\mathsf{P_A}$'s	$\mathsf{P_B}$'s	Total	$\mathsf{P_A}$'s	$\mathsf{P_B}$'s	Total
AES-128	0.207	0.003	0.210	0.214	0.214	0.428	1.908	1.358	3.266
SHA-128	1.138	0.003	1.141	1.146	1.146	2.292	10.429	7.430	17.859
SHA-256	2.772	0.002	2.774	2.784	2.784	5.568	25.367	18.080	43.446
Hamming Dist.	63.999	0.163	64.162	64.163	64.163	128.326	655.255	455.257	1,110.497
Integer Multi.	127.938	0.002	127.940	128.034	128.034	256.068	1170.087	834.178	2,004.265
Sorting	312.000	0.002	312.002	312.240	312.240	624.480	2862.491	2038.866	4,901.358

Communication Cost. Table 4 presents a comparison of per-party and total communication overhead between our protocol and both passively and actively secure 2PC protocols. In passively and actively secure GC-based 2PC protocols, $\mathsf{P_A}$ is the garbler while $\mathsf{P_B}$ is the evaluator. In passively secure GC-based 2PC protocols, $\mathsf{P_A}$ needs to generate and send a large number of garbled circuits to $\mathsf{P_B}$, with communication costs proportional to the number of AND gates. In contrast, most of $\mathsf{P_B}$'s tasks involve performing the GC evaluation locally without requiring extensive communication. As a result, the communication cost for $\mathsf{P_A}$ is significantly higher than that for $\mathsf{P_B}$. In actively secure GC-based 2PC protocols, although $\mathsf{P_A}$ acts as the garbler, the garbled circuits are generated jointly with $\mathsf{P_B}$. Consequently, the communication cost for $\mathsf{P_A}$ is higher than that for $\mathsf{P_B}$, though both costs remain within the same order of magnitude. In our protocol, $\mathsf{P_A}$ and $\mathsf{P_B}$ perform GC evaluations symmetrically and exchange data of nearly identical size during the verification and output progressive revelation phases, resulting in equal communication costs for both parties.

In terms of total communication, our protocol's communication cost is double that of the passively secure 2PC protocol, as it requires two symmetric GC evaluations to compute a circuit. However, compared to the actively secure 2PC protocol, our protocol achieves a 7.6−8.6× reduction in communication cost by eliminating complex verification overhead, highlighting its high communication efficiency.

Cost Breakdown of Our Low-Overhead Protocol Execution. We analyze our protocol by breaking down the execution cost of each component for `AES-128`, `SHA-128`, `SHA-256`, Hamming distance, integer multiplication, and sorting circuits. As previously outlined, our protocol is divided into three phases: evaluation, verification, and output progressive revelation. We measured the average wall-clock time for each phase of a single protocol execution in both LAN and WAN settings. As shown in Figure 3, circuit evaluation in the first phase accounts for 93.88% to 99.80% of the total runtime



(a) Running time for LAN (ms)

(b) Running time for WAN (ms)

Fig. 3: Cost breakdown of our low-overhead protocol execution in the LAN and WAN settings.

for most circuits. The verification and output progressive revelation phases we introduced contribute only a very small portion to the total runtime. This is because the evaluation phase incurs overhead proportional to the number of AND gates, while the verification and output progressive revelation phases scale linearly with the output length n_{O}. Since the output length n_{O} is generally much smaller than the number of AND gates in most circuits, our protocol achieves high efficiency.

Different Progressive Revelation Factors. Our protocol supports adjustable output progressive revelation, allowing the adversary's one-bit advantage to be extended to an arbitrary-bit advantage. This flexibility can reduce the number of communication rounds, making it valuable in scenarios where the size of evaluation result is large or network latency is high, thus achieving a trade-off between security and efficiency. To assess this, we conducted experiments to evaluate the impact of the progressive revelation factor on performance. Table 5 presents the average running time required to release each output bit, calculated by dividing the total running time of the output progressive revelation phase by the output size, with the revelation factor set to k. The reported data was obtained using the `AES-128` circuit. However, since the output progressive revelation phase of our protocol depends only on the output size and is independent of the specific circuit used, the results in Table 5 are generalizable and not tied to a particular circuit. As shown in the table, as k increases, the running time initially decreases rapidly and then plateaus in both LAN and WAN settings. The running time with $k = 1$ is 68.3× longer than with $k = 128$ in the LAN setting and 114.6× longer in the WAN setting. The impact on running time primarily stems from the reduction in the number of communication rounds in the protocol's output progressive revelation phase, which is calculated as $\lceil n_{\mathsf{O}}/k \rceil + 1$, where n_{O} is the output length of the circuits. Hence, as k increases, the number of communication rounds decreases significantly, resulting in a proportional reduction in network I/O time. This effect is particularly beneficial for communication-sensitive applications.

Table 5: The impact of different progressive revelation factors k on the protocol's running time, assuming the circuit's output length exceeds k.

Revelation factor	Running time for LAN	Running time for WAN
$k = 1$	119.594 $\times 10^{-3}$ ms	60.50 ms
$k = 8$	18.352 $\times 10^{-3}$ ms	7.503 ms
$k = 32$	4.563 $\times 10^{-3}$ ms	1.876 ms
$k = 64$	1.898 $\times 10^{-3}$ ms	0.938 ms
$k = 128$	1.750 $\times 10^{-3}$ ms	0.528 ms

7 Acknowledgements

We would like to express our sincere appreciation to the anonymous reviewers for their valuable comments. This work was supported in part by National Key Research and Development Program of China under Grant No. 2021ZD0112802, in part by National Natural Science Foundation

of China under Grant Nos. 62302194, 62472198, 62072215, 62250710682, 62332007, and U22B2028, in part by Guangzhou Basic and Applied Basic Research Foundation under Grant Nos. 2025A04J2146, 2024A03J0405, and 2024A04J3458, in part by Guangdong Basic and Applied Basic Research Foundation under Grant Nos. 2023B1515040020 and 2019B030302008, in part by Science and Technology Major Project of Tibetan Autonomous Region of China (No. XZ202201ZD0006G), in part by Open Research Fund of Machine Learning and Cyber Security Interdiscipline Research Engineering Center of Jiangsu Province (No. SDGC2131), in part by HKU-SCF FinTech Academy and Shenzhen-Hong Kong-Macao Science and Technology Plan Project (Category C Project: SGDX20210823103537030) and Theme-based Research Scheme T35-710/20-R, and in part by National Joint Engineering Research Center of Network Security Detection and Protection Technology, Guangdong Key Laboratory of Data Security and Privacy Preserving, Guangdong Hong Kong Joint Laboratory for Data Security and Privacy Protection, and Engineering Research Center of Trustworthy AI, Ministry of Education.

References

1. Abascal, J., Sereshgi, M.H.F., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Is the classical GMW paradigm practical? the case of non-interactive actively secure 2pc. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020. pp. 1591–1605. ACM (2020)
2. Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols (extended abstract). In: Ortiz, H. (ed.) Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA. pp. 503–513. ACM (1990)
3. Bellare, M., Hoang, V.T., Keelveedhi, S., Rogaway, P.: Efficient garbling from a fixed-key blockcipher. In: 2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013. pp. 478–492. IEEE Computer Society (2013)
4. Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012. pp. 784–796. ACM (2012)
5. Blum, M.: How to exchange (secret) keys. ACM Trans. Comput. Syst. **1**(2), 175–193 (1983)
6. Brüggemann, A., Hundt, R., Schneider, T., Suresh, A., Yalame, H.: FLUTE: fast and secure lookup table evaluations. In: 44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023. pp. 515–533. IEEE (2023)
7. Cui, H., Wang, X., Yang, K., Yu, Y.: Actively secure half-gates with minimum overhead under duplex networks. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part II. Lecture Notes in Computer Science, vol. 14005, pp. 35–67. Springer (2023)
8. Damgård, I.: Practical and provably secure release of a secret and exchange of signatures. J. Cryptol. **8**(4), 201–222 (1995)
9. Disser, Y., Günther, D., Schneider, T., Stillger, M., Wigandt, A., Yalame, H.: Breaking the size barrier: Universal circuits meet lookup tables. In: Guo, J., Steinfeld, R. (eds.) Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4-8, 2023, Proceedings, Part I. Lecture Notes in Computer Science, vol. 14438, pp. 3–37. Springer (2023)
10. Dittmer, S., Ishai, Y., Lu, S., Ostrovsky, R.: Authenticated garbling from simple correlations. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part IV. Lecture Notes in Computer Science, vol. 13510, pp. 57–87. Springer (2022)
11. Even, S., Goldreich, O., Lempel, A.: A randomized protocol for signing contracts. Commun. ACM **28**(6), 637–647 (1985)
12. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A.V. (ed.) Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA. pp. 218–229. ACM (1987)
13. Günther, D., Schmidt, J., Schneider, T., Yalame, H.: FLUENT: A tool for efficient mixed-protocol semi-private function evaluation. In: Annual Computer Security Applications Conference, ACSAC 2024, Honolulu, Hawaii, USA, December 9-13, 2024. pp. 1–14. ACM (2024)
14. Guo, C., Katz, J., Wang, X., Weng, C., Yu, Y.: Better concrete security for half-gates garbling (in the multi-instance setting). In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12171, pp. 793–822. Springer (2020)

15. Guo, C., Katz, J., Wang, X., Yu, Y.: Efficient and secure multiparty computation from fixed-key block ciphers. In: 2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020. pp. 825–841. IEEE (2020)
16. Hamacher, K., Kussel, T., Schneider, T., Tkachenko, O.: PEA: practical private epistasis analysis using MPC. In: Atluri, V., Pietro, R.D., Jensen, C.D., Meng, W. (eds.) Computer Security - ESORICS 2022 - 27th European Symposium on Research in Computer Security, Copenhagen, Denmark, September 26-30, 2022, Proceedings, Part III. Lecture Notes in Computer Science, vol. 13556, pp. 320–339. Springer (2022)
17. Hazay, C., Shelat, A., Venkitasubramaniam, M.: Going beyond dual execution: MPC for functions with efficient verification. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) Public-Key Cryptography - PKC 2020 - 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4-7, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12111, pp. 328–356. Springer (2020)
18. Hong, C., Katz, J., Kolesnikov, V., Lu, W., Wang, X.: Covert security with public verifiability: Faster, leaner, and simpler. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III. Lecture Notes in Computer Science, vol. 11478, pp. 97–121. Springer (2019)
19. Huang, Y., Katz, J., Evans, D.: Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution. In: IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA. pp. 272–284. IEEE Computer Society (2012)
20. Huang, Z., Lu, W., Wang, Y., Hong, C., Wei, T., Chen, W.: *Coral:* maliciously secure computation framework for packed and mixed circuits. In: Luo, B., Liao, X., Xu, J., Kirda, E., Lie, D. (eds.) Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024. pp. 810–824. ACM (2024)
21. Impagliazzo, R., Yung, M.: Direct minimum-knowledge computations. In: Pomerance, C. (ed.) Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings. Lecture Notes in Computer Science, vol. 293, pp. 40–51. Springer (1987)
22. Katz, J., Ranellucci, S., Rosulek, M., Wang, X.: Optimizing authenticated garbling for faster secure two-party computation. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III. Lecture Notes in Computer Science, vol. 10993, pp. 365–391. Springer (2018)
23. Kolesnikov, V., Mohassel, P., Riva, B., Rosulek, M.: Richer efficiency/security trade-offs in 2pc. In: Dodis, Y., Nielsen, J.B. (eds.) Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9014, pp. 229–259. Springer (2015)
24. Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations. Lecture Notes in Computer Science, vol. 5126, pp. 486–498. Springer (2008)
25. Leuven, K.: SCALE and MAMBA. `https://github.com/KULeuven-COSIC/SCALE-MAMBA/` (2018)
26. Liu, Y., Lai, J., Wang, Q., Qin, X., Yang, A., Weng, J.: Robust publicly verifiable covert security: Limited information leakage and guaranteed correctness with low overhead. In: Guo, J., Steinfeld, R. (eds.) Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4-8, 2023, Proceedings, Part I. Lecture Notes in Computer Science, vol. 14438, pp. 272–301. Springer (2023)
27. Mohassel, P., Franklin, M.K.: Efficiency tradeoffs for malicious two-party computation. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings. Lecture Notes in Computer Science, vol. 3958, pp. 458–473. Springer (2006)
28. Mohassel, P., Riva, B.: Garbled circuits checking garbled circuits: More efficient and secure two-party computation. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II. Lecture Notes in Computer Science, vol. 8043, pp. 36–53. Springer (2013)
29. Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: Feldman, S.I., Wellman, M.P. (eds.) Proceedings of the First ACM Conference on Electronic Commerce (EC-99), Denver, CO, USA, November 3-5, 1999. pp. 129–139. ACM (1999)
30. Rindal, P., Rosulek, M.: Faster malicious 2-party secure computation with online/offline dual execution. In: Holz, T., Savage, S. (eds.) 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016. pp. 297–314. USENIX Association (2016)

31. Rindal, P., Rosulek, M.: Malicious-secure private set intersection via dual execution. In: Thuraisingham, B., Evans, D., Malkin, T., Xu, D. (eds.) Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017. pp. 1229–1242. ACM (2017)
32. Rosulek, M., Roy, L.: Three halves make a whole? beating the half-gates lower bound for garbled circuits. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12825, pp. 94–124. Springer (2021)
33. Wang, X., Malozemoff, A.J., Katz, J.: EMP-toolkit: Efficient MultiParty computation toolkit. `https://github.com/emp-toolkit` (2016)
34. Wang, X., Ranellucci, S., Katz, J.: Authenticated garbling and efficient maliciously secure two-party computation. In: Thuraisingham, B., Evans, D., Malkin, T., Xu, D. (eds.) Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017. pp. 21–37. ACM (2017)
35. Yang, K., Wang, X., Zhang, J.: More efficient MPC from improved triple generation and authenticated garbling. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020. pp. 1627–1646. ACM (2020)
36. Yao, A.C.: Protocols for secure computations (extended abstract). In: 23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982. pp. 160–164. IEEE Computer Society (1982)
37. Yao, A.C.: How to generate and exchange secrets (extended abstract). In: 27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986. pp. 162–167. IEEE Computer Society (1986)
38. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole - reducing data transfer in garbled circuits using half gates. In: Oswald, E., Fischlin, M. (eds.) 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. Lecture Notes in Computer Science, vol. 9057, pp. 220–250. Springer (2015)
39. Zhang, W., Guo, X., Yang, K., Zhu, R., Yu, Y., Wang, X.: Efficient actively secure DPF and ram-based 2PC with one-bit leakage. In: IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, May 19-23, 2024. pp. 561–577. IEEE (2024)

A Proof of Security

The rest of this section is devoted to a proof of the following theorem:

Theorem 1. *The protocol $\Pi_{\mathsf{OneBitAdv}}$ along with a secure projective garbling scheme that is point-permute-freeXOR-compatible securely realizes functionality $\mathcal{F}_{\mathsf{OneBitAdv}}$ against* PPT *malicious (rushing) adversaries in the ($\mathcal{F}_{\mathsf{Com}}$, $\mathcal{F}_{\mathsf{OT}}$, $\mathcal{F}_{\mathsf{OLE}}$, $\mathcal{F}_{\mathsf{Rand}}$, $\mathcal{F}_{\mathsf{Eq}}$)-hybrid world.*

Proof. Without loss of generality, we assume that P_B is honest, while P_A is corrupted by the adversary $\mathcal{A}$. The roles of P_A and P_B in protocol $\Pi_{\mathsf{OneBitAdv}}$ are symmetric, so a similar proof applies when P_B is corrupted.

For an adversary $\mathcal{A}$ corrupting P_A in the ($\mathcal{F}_{\mathsf{Com}}$, $\mathcal{F}_{\mathsf{OT}}$, $\mathcal{F}_{\mathsf{OLE}}$, $\mathcal{F}_{\mathsf{Rand}}$, $\mathcal{F}_{\mathsf{Eq}}$)-hybrid world, we construct a simulator $\mathcal{S}$, which plays the role of P_B and runs $\mathcal{A}$ as a subroutine with auxiliary input z, interacting with $\mathcal{F}_{\mathsf{OneBitAdv}}$ in the ideal world. The simulation has been given in Figure 2.

We now proceed to prove that the joint distribution of the view of $\mathcal{A}$ and the output of P_B in the ideal world is computationally indistinguishable from the joint distribution of the view of $\mathcal{A}$ and the output of P_B in the real protocol execution. We prove this by defining a sequence of experiments, where the output of each consists of the view of $\mathcal{A}$ and the output of P_B, and showing that the output of each is computationally indistinguishable from the output of the subsequent one.

Expt$_0$. This experiment represents the ideal-world execution between the simulator $\mathcal{S}$ and the honest party P_B, holding the input x_B. Both interact with $\mathcal{F}_{\mathsf{OneBitAdv}}$. We inline the actions of $\mathcal{S}$, $\mathcal{F}_{\mathsf{OneBitAdv}}$, and P_B, and rewrite the experiment as follows.

1 Sample $\Delta_\mathsf{B} \leftarrow\!\!\$\ \mathbb{F}_2^\kappa$, such that $\mathsf{lsb}(\Delta_\mathsf{B}) = 1$.
2 Compute

$$(\mathsf{GC}', X', z) \leftarrow \mathcal{S}_{\mathsf{Gb}}(1^\kappa, \mathcal{C})$$

where $X' = \{X'_i\}_{i\in[n_\mathsf{O}]}$. Let $Y' = \{Y'_i\}_{i\in[n_\mathsf{O}]} = \mathsf{Ev}(\mathsf{GC}', X')$. For each output wire $w = \mathsf{out}(i)$, $\mathcal{S}$ defines $\hat{z}'_w = \mathsf{lsb}(Y'_i)$.

Acting as $\mathcal{F}_{\mathsf{Com}}$, inform the adversary $\mathcal{A}$ that the honest party P_B has committed his garbled circuit to $\mathcal{F}_{\mathsf{Com}}$. Subsequently, obtain the garbled circuit GC from $\mathcal{A}$.

3 If P_A is the receiver in $\mathcal{F}_{\mathsf{OT}}$, receive x_A from $\mathcal{A}$, send $\{X'_i\}_{i\in[n_\mathsf{A}]}$ to $\mathcal{A}$, and store x_A. If P_A is the sender in $\mathcal{F}_{\mathsf{OT}}$, receive $\{(X_{i,0}, X_{i,1})\}_{i\in[n_\mathsf{A}+1,n]}$ from $\mathcal{A}$.

4 Send $\{X'_i\}_{i\in[n_\mathsf{A}+1,n]}$ to $\mathcal{A}$ and receive $\{X_i\}_{i\in[n_\mathsf{A}]}$ from $\mathcal{A}$.

5 Acting as $\mathcal{F}_{\mathsf{Com}}$, open GC' to $\mathcal{A}$.

6 As the receiver of $\mathcal{F}_{\mathsf{OLE}}$, receive L_0 and $\hat{\Delta}_\mathsf{A}$ from $\mathcal{A}$. As the sender, receive a_0 from $\mathcal{A}$ and return $A_{0,2} \leftarrow\!\!\$\, \mathbb{F}_2^\kappa$ to $\mathcal{A}$. Set $\mathsf{L}'_0 := A_{0,2} \oplus a_0\Delta_\mathsf{B}$.

7 For $i \in [n_\mathsf{O}]$, define the Boolean function g_i (resp. g'_i) with fixed input x_A and variable $x_\mathsf{B} \in \{0,1\}^{n_\mathsf{B}}$ as follows:
   (a) Let $X_i := X_{i,x_\mathsf{B}[i-n_\mathsf{A}]}$ for $i \in [n_\mathsf{A}+1, n]$.
   (b) Compute $Y := \mathsf{Ev}(\mathsf{GC}, X)$, where $X = \{X_i\}_{i\in[n]}$.
   (c) Define $B_{i,1} := Y_i$ and $b_{i,1} := \hat{z}_w = \mathsf{lsb}(Y_i)$ for $w = \mathsf{out}(i)$.
   (d) Compute $y := \mathcal{C}(x_\mathsf{A}, x_\mathsf{B})$, where $y = (y_1, \cdots, y_{n_\mathsf{O}}) \in \{0,1\}^{n_\mathsf{O}}$.
   (e) Let $\lambda'_w := y_i \oplus \hat{z}'_w$, and define $b_{i,2} := \lambda'_w$ for $w = \mathsf{out}(i)$.
   (f) Set $B_{i,2} := Y'_i \oplus y_i\Delta_\mathsf{B}$ and $B_i := B_{i,1} \oplus B_{i,2}$.
   (g) Output $\alpha_i := B_i \oplus b_{i,1}\hat{\Delta}_\mathsf{A} \oplus b_{i,1}\Delta_\mathsf{B}$ (resp. $\alpha'_i := B_i \oplus b_{i,2}\hat{\Delta}_\mathsf{A} \oplus b_{i,2}\Delta_\mathsf{B}$).
   Compute $\alpha_i := g_i(x_\mathsf{A}, x_\mathsf{B})$ and $\alpha'_i := g'_i(x_\mathsf{A}, x_\mathsf{B})$ for $i \in [n_\mathsf{O}]$, pick random values $\{r_i, r'_i\}_{i\in[n_\mathsf{O}]}$, and forward these random values to $\mathcal{A}$ with respect to $\mathcal{F}_{\mathsf{Rand}}$.
   Randomly select $b \leftarrow\!\!\$\, \mathbb{F}_2^\kappa$ and send it to $\mathcal{A}$. Meanwhile, receive $a \in \mathbb{F}_2^\kappa$ from $\mathcal{A}$.

8 Receive $\hat{\beta} \in \mathbb{F}_2^\kappa$ from $\mathcal{A}$ with respect to $\mathcal{F}_{\mathsf{Eq}}$. Compute $\beta := \hat{\beta} \oplus b\hat{\Delta}_\mathsf{A} \oplus \mathsf{L}_0 \oplus \mathsf{L}'_0 \oplus a\Delta_\mathsf{B}$, and then check if $\beta = \left(\sum_{i\in[n_\mathsf{O}]} r_i\alpha_i\right) \oplus \left(\sum_{i\in[n_\mathsf{O}]} r'_i\alpha'_i\right)$. If the equation does not hold, simulate rejection by honest P_B. Otherwise, proceed to the next step.

9 Compute $y := \mathcal{C}(x_\mathsf{A}, x_\mathsf{B})$. For $i \in [n_\mathsf{O}]$, proceed as follows.
   (a) Compute $b_{i,2} = \lambda'_w := y_i \oplus \hat{z}'_w$ for $w = \mathsf{out}(i)$ and send it to $\mathcal{A}$. Simultaneously, receive $a_{i,1}$ from $\mathcal{A}$.
   (b) Receive $\tilde{\alpha}_i$ with respect to $\mathcal{F}_{\mathsf{Eq}}$. Then compute $\hat{b}_{i,1} := y_i \oplus a_{i,1}$ and check if $\alpha_i \oplus \hat{b}_{i,1}\Delta_\mathsf{B} \oplus \hat{b}_{i,1}\hat{\Delta}_\mathsf{A} = \alpha'_i \oplus b_{i,2}\Delta_\mathsf{B} \oplus b_{i,2}\hat{\Delta}_\mathsf{A}$. If the equation does not hold, simulate rejection by P_B.
   Then check if $\tilde{\alpha}_i = \alpha_i \oplus a_{i,1}\Delta_\mathsf{B} \oplus a_{i,1}\hat{\Delta}_\mathsf{A} \oplus y_i\hat{\Delta}_\mathsf{A}$. If the equation does not hold, simulate rejection by P_B. Otherwise, output whatever $\mathcal{A}$ outputs and continue.

Expt$_1$. Since each g_i and g'_i perform similar computation in steps 7a–7g, redundant calculations can be combined. Step 7 of the previous experiment can be modified as follows:

7 Pick random $\{r_i, r'_i\}_{i\in[n_\mathsf{O}]}$ and forward them to $\mathcal{A}$ as $\mathcal{F}_{\mathsf{Rand}}$.
   Follow the procedure below to compute α_i and α'_i for $i \in [n_\mathsf{O}]$.
   (a) Let $X_i := X_{i,x_\mathsf{B}[i-n_\mathsf{A}]}$ for $i \in [n_\mathsf{A}+1, n]$.
   (b) Compute $Y := \mathsf{Ev}(\mathsf{GC}, X)$, where $X = \{X_i\}_{i\in[n]}$.
   (c) Define $B_{i,1} := Y_i$ and $b_{i,1} := \hat{z}_w = \mathsf{lsb}(Y_i)$ for $w = \mathsf{out}(i)$ and $i \in [n_\mathsf{O}]$.
   (d) Compute $y := \mathcal{C}(x_\mathsf{A}, x_\mathsf{B})$, where $y = (y_1, \cdots, y_{n_\mathsf{O}}) \in \{0,1\}^{n_\mathsf{O}}$.
   (e) Let $\lambda'_w := y_i \oplus \hat{z}'_w$, and define $b_{i,2} := \lambda'_w$ for $w = \mathsf{out}(i)$ and $i \in [n_\mathsf{O}]$.
   (f) Set $B_{i,2} := Y'_i \oplus y_i\Delta_\mathsf{B}$ and $B_i := B_{i,1} \oplus B_{i,2}$ for $i \in [n_\mathsf{O}]$.
   (g) Let $\alpha_i := B_i \oplus b_{i,1}\hat{\Delta}_\mathsf{A} \oplus b_{i,1}\Delta_\mathsf{B}$ and $\alpha'_i := B_i \oplus b_{i,2}\hat{\Delta}_\mathsf{A} \oplus b_{i,2}\Delta_\mathsf{B}$ for $i \in [n_\mathsf{O}]$.
   Randomly select $b \leftarrow\!\!\$\, \mathbb{F}_2^\kappa$ and send it to $\mathcal{A}$. Meanwhile, receive $a \in \mathbb{F}_2^\kappa$ from $\mathcal{A}$.

It is easy to verify that the outputs of **Expt**$_1$ and **Expt**$_0$ are identically distributed.

Expt$_2$. Move the computation of $y := \mathcal{C}(x_\mathsf{A}, x_\mathsf{B})$ from Step 7 to Step 3, with the same computation of y in Step 9 eliminated. Since the computation of y based on x_A, x_B, and $\mathcal{C}$ is deterministic, this rearrangement does not alter the output distribution. Additionally, the assignment $X_i := X_{i,x_\mathsf{B}[i-n_\mathsf{A}]}$ for $i \in [n_\mathsf{A}+1, n]$ is moved from Step 7 to Step 3. Moreover, computations including $Y := \mathsf{Ev}(\mathsf{GC}, X)$, $B_{i,2} := Y'_i \oplus y_i\Delta_\mathsf{B}$, $\lambda'_w := y_i \oplus \hat{z}'_w$, $b_{i,2} := \lambda'_w$, $B_{i,1} := Y_i$, $b_{i,1} := \hat{z}_w = \mathsf{lsb}(Y_i)$, and $B_i := B_{i,1} \oplus B_{i,2}$ for $w = \mathsf{out}(i)$ and $i \in [n_\mathsf{O}]$, are moved from Step 7 to Step 5. We also note that $b_{i,2}$ does not need to be recomputed in Step 9, as it is already computed (in Step 5 of this experiment).

It is evident that the outputs of **Expt**$_2$ and **Expt**$_1$ are identically distributed.

Expt$_3$. Step 6 of the previous experiment is modified as follows.

6 Use $\mathcal{F}_{\mathsf{OLE}}$ to receive L_0 and $\hat{\Delta}_\mathsf{A}$ from $\mathcal{A}$. Sample $\mathsf{L}'_0 \leftarrow\!\!\$\, \mathbb{F}_2^\kappa$. As the sender of $\mathcal{F}_{\mathsf{OLE}}$, receive a_0 from $\mathcal{A}$, compute $A_{0,2} := \mathsf{L}'_0 \oplus a_0\Delta_\mathsf{B}$, and send it to $\mathcal{A}$.

Since L'_0 is uniquely defined by the randomly generated $A_{0,2} \in \mathbb{F}_2^\kappa$ in the previous experiment, altering the order in which $A_{0,2}$ and L'_0 are generated does not impact the output distribution of the experiment. Consequently, the outputs of $\mathbf{Expt}_3$ and $\mathbf{Expt}_2$ remain identically distributed.
$\mathbf{Expt}_4$. Step 3 in the previous experiment is modified as follows.

> 3 If $\mathsf{P_A}$ is the receiver in $\mathcal{F}_{\mathsf{OT}}$, receive x_A from $\mathcal{A}$ and send $\{X'_i\}_{i\in[n_\mathsf{A}]}$ to $\mathcal{A}$. Compute $y := \mathcal{C}(x_\mathsf{A}, x_\mathsf{B})$. If $\mathsf{P_A}$ is the sender, use $\mathcal{F}_{\mathsf{OT}}$ with input x_B to obtain $X_i := X_{i,x_\mathsf{B}[i-n_\mathsf{A}]}$ for $i \in [n_\mathsf{A}+1, n]$.

Since values $X_{i,1-x_\mathsf{B}[i-n_\mathsf{A}]}$ are not used in the experiment, executing $\mathcal{F}_{\mathsf{OT}}$ honestly does not change the output of the experiment.

Step 6 is also modified as follows.

> 6 Pick $b_0 \leftarrow\!\!\$\ \mathbb{F}_2^\kappa$. As the receiver, use $\mathcal{F}_{\mathsf{OLE}}$ to receive $B_{0,1} := \mathsf{L}_0 \oplus b_0\hat{\Delta}_\mathsf{A}$, where L_0 and $\hat{\Delta}_\mathsf{A}$ are from $\mathcal{A}$. Sample $\mathsf{L}'_0 \leftarrow\!\!\$\ \mathbb{F}_2^\kappa$. As the sender, use $\mathcal{F}_{\mathsf{OLE}}$ with input L'_0 and Δ_B, sending $A_{0,2} := \mathsf{L}'_0 \oplus a_0\Delta_\mathsf{B}$ based on $\mathcal{A}$'s input a_0 to $\mathcal{A}$. Let $B_{0,2} := \mathsf{L}'_0 \oplus b_0\Delta_\mathsf{B}$ and $B_0 := B_{0,1} \oplus B_{0,2}$.

In this modified step, $\mathcal{F}_{\mathsf{OLE}}$ is honestly executed in the sender role. Additionally, added $B_{0,1}$, $B_{0,2}$ and B_0 are values unused elsewhere in the protocol. Therefore, these modifications do not affect the experiment's output.

In Step 7, the value b sent to $\mathcal{A}$ is now computed as $b := b_0 \oplus \left(\sum_{i\in[n_\mathsf{O}]} r_i b_{i,1}\right) \oplus \left(\sum_{i\in[n_\mathsf{O}]} r'_i b_{i,2}\right)$, and we rewrite it as follows.

> 7 Use $\mathcal{F}_{\mathsf{Rand}}$ to choose random $\{r_i, r'_i\}_{i\in[n_\mathsf{O}]}$ and forward them to $\mathcal{A}$.
> Compute $b := b_0 \oplus \left(\sum_{i\in[n_\mathsf{O}]} r_i b_{i,1}\right) \oplus \left(\sum_{i\in[n_\mathsf{O}]} r'_i b_{i,2}\right)$ and send it to $\mathcal{A}$. Meanwhile, receive $a \in \mathbb{F}_2^\kappa$ from $\mathcal{A}$.
> Let $\alpha_i := B_i \oplus b_{i,1}\hat{\Delta}_\mathsf{A} \oplus b_{i,1}\Delta_\mathsf{B}$ and $\alpha'_i := B_i \oplus b_{i,2}\hat{\Delta}_\mathsf{A} \oplus b_{i,2}\Delta_\mathsf{B}$ for $i \in [n_\mathsf{O}]$.

Since b_0 is randomly selected and is not used elsewhere, b maintains the same distribution as before. The outputs of $\mathbf{Expt}_4$ and $\mathbf{Expt}_3$ are thus identically distributed.
$\mathbf{Expt}_5$. In this experiment, Step 8 from the previous experiment is modified as follows.

> 8 Receive $\hat{\beta} \in \mathbb{F}_2^\kappa$ from $\mathcal{A}$ with respect to $\mathcal{F}_{\mathsf{Eq}}$. Verify if $\hat{\beta} = B_0 \oplus \left(\sum_{i\in[n_\mathsf{O}]} r_i B_i\right) \oplus \left(\sum_{i\in[n_\mathsf{O}]} r'_i B_i\right) \oplus (a \oplus b)\Delta_\mathsf{B}$. If this equation does not hold, simulate a rejection by the honest $\mathsf{P_B}$. Otherwise, proceed to the next step.

In the previous experiment, we compute $\beta := \hat{\beta} \oplus b\hat{\Delta}_\mathsf{A} \oplus \mathsf{L}_0 \oplus \mathsf{L}'_0 \oplus a\Delta_\mathsf{B}$ and check if $\beta = \left(\sum_{i\in[n_\mathsf{O}]} r_i\alpha_i\right) \oplus \left(\sum_{i\in[n_\mathsf{O}]} r'_i\alpha'_i\right)$. This is equivalent to verifying that

$$
\begin{aligned}
\hat{\beta} &= \left(\sum_{i\in[n_\mathsf{O}]} r_i\alpha_i\right) \oplus \left(\sum_{i\in[n_\mathsf{O}]} r'_i\alpha'_i\right) \oplus b\hat{\Delta}_\mathsf{A} \oplus \mathsf{L}_0 \oplus \mathsf{L}'_0 \oplus a\Delta_\mathsf{B} \\
&= \left(\sum_{i\in[n_\mathsf{O}]} r_i \left(B_i \oplus b_{i,1}\hat{\Delta}_\mathsf{A} \oplus b_{i,1}\Delta_\mathsf{B}\right)\right) \\
&\quad \oplus \left(\sum_{i\in[n_\mathsf{O}]} r'_i \left(B_i \oplus b_{i,2}\hat{\Delta}_\mathsf{A} \oplus b_{i,2}\Delta_\mathsf{B}\right)\right) \\
&\quad \oplus b\hat{\Delta}_\mathsf{A} \oplus b\Delta_\mathsf{B} \oplus \mathsf{L}_0 \oplus \mathsf{L}'_0 \oplus a\Delta_\mathsf{B} \oplus b\Delta_\mathsf{B} \\
&= \left(\sum_{i\in[n_\mathsf{O}]} r_i B_i\right) \oplus \left(\sum_{i\in[n_\mathsf{O}]} r'_i B_i\right) \oplus b_0\hat{\Delta}_\mathsf{A} \oplus b_0\Delta_\mathsf{B} \\
&\quad \oplus \mathsf{L}_0 \oplus \mathsf{L}'_0 \oplus (a \oplus b)\Delta_\mathsf{B} \\
&= B_0 \oplus \left(\sum_{i\in[n_\mathsf{O}]} r_i B_i\right) \oplus \left(\sum_{i\in[n_\mathsf{O}]} r'_i B_i\right) \oplus (a \oplus b)\Delta_\mathsf{B}\,,
\end{aligned}
$$

where we use the fact that $b_0 = b \oplus \left(\sum_{i\in[n_\mathsf{O}]} r_i b_{i,1}\right) \oplus \left(\sum_{i\in[n_\mathsf{O}]} r'_i b_{i,2}\right)$, $B_{0,1} = \mathsf{L}_0 \oplus b_0\hat{\Delta}_\mathsf{A}$, $B_{0,2} = \mathsf{L}'_0 \oplus b_0\Delta_\mathsf{B}$, and $B_0 = B_{0,1} \oplus B_{0,2}$. Therefore, the outputs of $\mathbf{Expt}_5$ and $\mathbf{Expt}_4$ are identically distributed.

$\mathbf{Expt}_6$. Step 9 in the previous experiment is modified as follows.

> 9 For $i \in [n_\mathsf{O}]$, follow the procedure below.
> (a) Send $b_{i,2}$ to $\mathcal{A}$. Meanwhile, receive $a_{i,1}$ from $\mathcal{A}$.
> (b) Receive $\tilde{\alpha}_i$ with respect to $\mathcal{F}_{\mathsf{Eq}}$. Then check if $b_{i,1} \oplus a_{i,1} = y_i$. If the equation does not hold, simulate rejection by P_B.
> Then check if $\tilde{\alpha}_i = B_i \oplus y_i\Delta_\mathsf{B}$. If the equation does not hold, simulate rejection by P_B. Otherwise, output whatever $\mathcal{A}$ outputs and continue.

In the previous experiment, we begin by verifying if $\alpha_i \oplus \hat{b}_{i,1}\Delta_\mathsf{B} \oplus \hat{b}_{i,1}\hat{\Delta}_\mathsf{A} = \alpha'_i \oplus b_{i,2}\Delta_\mathsf{B} \oplus b_{i,2}\hat{\Delta}_\mathsf{A}$, where $\hat{b}_{i,1} := y_i \oplus a_{i,1}$, $\alpha_i = B_i \oplus b_{i,1}\Delta_\mathsf{B} \oplus b_{i,1}\hat{\Delta}_\mathsf{A}$, and $\alpha'_i = B_i \oplus b_{i,2}\Delta_\mathsf{B} \oplus b_{i,2}\hat{\Delta}_\mathsf{A}$. This is equivalent to checking if

$$\begin{aligned} B_i \oplus b_{i,1}\Delta_\mathsf{B} \oplus b_{i,1}\hat{\Delta}_\mathsf{A} \oplus (y_i \oplus a_{i,1})\Delta_\mathsf{B} \oplus (y_i \oplus a_{i,1})\hat{\Delta}_\mathsf{A} \\ = B_i \oplus b_{i,2}\Delta_\mathsf{B} \oplus b_{i,2}\hat{\Delta}_\mathsf{A} \oplus b_{i,2}\Delta_\mathsf{B} \oplus b_{i,2}\hat{\Delta}_\mathsf{A} \end{aligned}$$

and thus checking

$$(b_{i,1} \oplus a_{i,1} \oplus y_i)(\Delta_\mathsf{B} \oplus \hat{\Delta}_\mathsf{A}) = 0\,.$$

Since Δ_B is randomly chosen and GC' is simulated, we have $\Delta_\mathsf{B} \neq \hat{\Delta}_\mathsf{A}$ except with negligible probability. Hence, verifying $b_{i,1} \oplus a_{i,1} = y_i$ is sufficient.

Furthermore, if $b_{i,1} \oplus a_{i,1} = y_i$, then checking $\tilde{\alpha}_i = \alpha_i \oplus a_{i,1}\Delta_\mathsf{B} \oplus a_{i,1}\hat{\Delta}_\mathsf{A} \oplus y_i\hat{\Delta}_\mathsf{A}$ is equivalent to verifying

$$\begin{aligned} \tilde{\alpha}_i &= B_i \oplus b_{i,1}\Delta_\mathsf{B} \oplus b_{i,1}\hat{\Delta}_\mathsf{A} \oplus a_{i,1}\Delta_\mathsf{B} \oplus a_{i,1}\hat{\Delta}_\mathsf{A} \oplus y_i\hat{\Delta}_\mathsf{A} \\ &= B_i \oplus y_i\Delta_\mathsf{B} \end{aligned}$$

Therefore, the output of $\mathbf{Expt}_6$ and $\mathbf{Expt}_5$ are identically distributed.

Since now α_i and α'_i are no longer required in this experiment, they can be safely removed.

$\mathbf{Expt}_7$. In this experiment, the functionality $\mathcal{F}_{\mathsf{Com}}$ in Step 2 and 4 is executed honestly as in the real world. Since GC and GC' are not used before Step 5, this modification does not change the output distribution. Additionally, in Step 7, compute $B := B_0 \oplus \left(\sum_{i\in[n_\mathsf{O}]} r_i B_i\right) \oplus \left(\sum_{i\in[n_\mathsf{O}]} r'_i B_i\right)$ and honestly execute $\mathcal{F}_{\mathsf{Eq}}$ in Step 8 to verify if $\hat{\beta} = B \oplus (a \oplus b)\Delta_\mathsf{B}$. It is straightforward to verify that the outputs of $\mathbf{Expt}_7$ and $\mathbf{Expt}_6$ remain identically distributed.

$\mathbf{Expt}_8$. Step 2 to Step 5 of the previous experiment is modified as follows.

> 2 Compute
> $$(\mathsf{GC}', e', \mathsf{dk}', d', D') \leftarrow \mathsf{Gb}(1^\kappa, \mathcal{C}; \Delta_\mathsf{B})\,,$$
> where $e' = \left\{(X'_{i,0}, X'_{i,1})\right\}$. Here dk' can be used to derive the point-permute bit λ'_w's and output-wire label $\mathsf{L}'_{w,0}$'s for each wire $w \in \mathcal{W}_\mathsf{O}$, respectively. Use $\mathcal{F}_{\mathsf{Com}}$ to commit GC'.
> 3 If P_A is the receiver in $\mathcal{F}_{\mathsf{OT}}$, send $\left\{X'_{i,x_\mathsf{A}[i]}\right\}_{i\in[n_\mathsf{A}]}$ with respect to $\mathcal{A}$'s input x_A to $\mathcal{A}$. If P_A is the sender, use $\mathcal{F}_{\mathsf{OT}}$ to obtain $X_i := X_{i,x_\mathsf{B}[i-n_\mathsf{A}]}$ for $i \in [n_\mathsf{A}+1, n]$. Compute $y := \mathcal{C}(x_\mathsf{A}, x_\mathsf{B})$.
> 4 Send $\left\{X'_i = X'_{i,x_\mathsf{B}[i-n_\mathsf{A}]}\right\}_{i\in[n_\mathsf{A}+1,n]}$ to $\mathcal{A}$. Receive $\{X_i\}_{i\in[n_\mathsf{A}]}$ from $\mathcal{A}$.
> 5 Open GC' to $\mathcal{A}$ via $\mathcal{F}_{\mathsf{Com}}$. Learn the garbled circuit GC generated by $\mathcal{A}$ from $\mathcal{F}_{\mathsf{Com}}$. Compute $Y := \mathsf{Ev}(\mathsf{GC}, X)$, where $X = \{X_i\}_{i\in[n]}$. Define $B_{i,1} := Y_i$ and $b_{i,1} := \hat{z}_w = \mathsf{lsb}(Y_i)$ for $w = \mathsf{out}(i)$ and $i \in [n_\mathsf{O}]$.
> Set $B_{i,2} := \mathsf{L}'_{w,0} \oplus \lambda'_w\Delta_\mathsf{B}$ and $b_{i,2} := \lambda'_w$ for $w = \mathsf{out}(i)$ and $i \in [n_\mathsf{O}]$. Let $B_i := B_{i,1} \oplus B_{i,2}$.

Since the garbling scheme achieves equivocable privacy, replacing the garbled circuits simulated by $(\mathcal{S}_{\mathsf{Gb}}, \mathcal{S}'_{\mathsf{Gb}})$ by an honestly generated garbled circuit only incurs a negligible difference in the output distribution. Note that in the previous experiment, we set $B_{i,2} := Y'_i \oplus y_i\Delta_\mathsf{B}$. Since

$$\begin{aligned} B_{i,2} = Y'_i \oplus y_i\Delta_\mathsf{B} &= \mathsf{L}'_{w,\hat{z}'_w} \oplus y_i\Delta_\mathsf{B} \\ &= \mathsf{L}'_{w,0} \oplus \hat{z}'_w\Delta_\mathsf{B} \oplus y_i\Delta_\mathsf{B} = \mathsf{L}'_{w,0} \oplus \lambda'_w\Delta_\mathsf{B} \end{aligned}$$

Computing $B_{i,2} = \mathsf{L}'_{w,0} \oplus \lambda'_w\Delta_\mathsf{B}$ derive the same $B_{i,2}$ as $B_{i,2} = Y'_i \oplus y_i\Delta_\mathsf{B}$. Therefore, the output distribution of $\mathbf{Expt}_8$ is computationally indistinguishable from that of $\mathbf{Expt}_7$.

$\mathbf{Expt}_9$. In this experiment, we modify Step 9 in the previous experiment as follows.

9 For $i \in [n_{\mathsf{O}}]$, follow the procedure below.
   (a) Send $b_{i,2}$ to $\mathcal{A}$. Meanwhile, receive $a_{i,1}$ from $\mathcal{A}$.
   (b) Receive $\tilde{\alpha}_i$ with respect to $\mathcal{F}_{\mathsf{Eq}}$. Then check if $\tilde{\alpha}_i = B_i \oplus (a_{i,1} \oplus b_{i,1})\Delta_{\mathsf{B}}$. If the equation does not hold, simulate rejection by honest P_{B}. Otherwise, output whatever $\mathcal{A}$ outputs and continue.

Since the value y is no longer used in this experiment, we can remove the assignment $y := \mathcal{C}(x_{\mathsf{A}}, x_{\mathsf{B}})$ in Step 3.

If $a_{i,1} \oplus b_{i,1} = y_i$, where $y = \mathcal{C}(x_{\mathsf{A}}, x_{\mathsf{B}})$ as computed in the previous experiment, then verifying if $\tilde{\alpha}_i = B_i \oplus y_i\Delta_{\mathsf{B}}$ is equivalent to verifying if $\tilde{\alpha}_i = B_i \oplus (a_{i,1} \oplus b_{i,1})\Delta_{\mathsf{B}}$. Thus, the output distribution of $\mathbf{Expt}_9$ is identical to that of $\mathbf{Expt}_8$.

We proceed to show that if $a_{i,1} \oplus b_{i,1} \neq y_i$, then $\tilde{\alpha}_i \neq B_i \oplus (a_{i,1} \oplus b_{i,1})\Delta_{\mathsf{B}}$ except with negligible probability. Consequently, P_{A} will reject $\mathcal{A}$ as in the previous experiment.

Define $\bar{y}_i = a_{i,1} \oplus b_{i,1} = y_i \oplus 1$. If $a_{i,1} \oplus b_{i,1} \neq y_i$ and $\tilde{\alpha}_i = B_i \oplus (a_{i,1} \oplus b_{i,1})\Delta_{\mathsf{B}}$, then the value $\tilde{\alpha}_i$ provided by $\mathcal{A}$ satisfies

$$\begin{aligned}
\tilde{\alpha}_i &= B_i \oplus (a_{i,1} \oplus b_{i,1})\Delta_{\mathsf{B}} = B_{i,1} \oplus B_{i,2} \oplus \bar{y}_i\Delta_{\mathsf{B}} \\
&= Y_i \oplus (\mathsf{L}'_{w,0} \oplus \lambda'_w\Delta_{\mathsf{B}}) \oplus \bar{y}_i\Delta_{\mathsf{B}} \\
&= Y_i \oplus \mathsf{L}'_{w,0} \oplus (\lambda'_w \oplus \bar{y}_i)\Delta_{\mathsf{B}} \\
&= Y_i \oplus \mathsf{L}'_{w,\hat{z}'_w \oplus 1}
\end{aligned}$$

for $w = \mathsf{out}(i)$, where we use the fact that $\lambda'_w \oplus \bar{y}_i = \hat{z}'_w \oplus 1$ in the last equation. Since $\mathcal{A}$ knows both $\tilde{\alpha}_i$ and Y_i, it can derive the output-wire label $\mathsf{L}'_{w,\hat{z}'_w \oplus 1}$ of the garbled circuit GC generated by the honest P_{B}. This result contradicts the authenticity against one-bit leakage property of the secure garbling scheme. Therefore, the output of $\mathbf{Expt}_9$ is indistinguishable from that of $\mathbf{Expt}_8$.

As $\mathbf{Expt}_9$ corresponds to a real-world execution of the protocol, this concludes the proof.

□