Authenticated Garbling and Communication-Efficient, Constant-Round, Secure Two-Party Computation

Jonathan Katz
University of Maryland
jkatz@cs.umd.edu

Samuel Ranellucci
University of Maryland
George Mason University
samuel@umd.edu

Xiao Wang
University of Maryland
wangxiao@cs.umd.edu

Abstract

We propose a simple and efficient framework for obtaining communication-efficient, constant-round protocols for (malicious) secure two-party computation. Our framework uses a preprocessing phase to generate authentication information for the two parties; in the online phase, this information is used to construct a *single* "authenticated" garbled circuit which is transmitted and evaluated. We also discuss various instantiations of our framework:

- The preprocessing phase can be instantiated efficiently using, e.g., TinyOT. Using this approach with our improvements to TinyOT, we obtain a protocol in which secure evaluation of an AES circuit (at 128-bit computational security and 40-bit statistical security) uses roughly 6 MB of communication in total. Most of the communication is circuit independent. A single execution of our protocol performs even better than the best previous work supporting circuit-independent preprocessing when amortized over 1024 executions.
- If the preprocessing phase is instantiated using the IPS compiler, we obtain a constant-round protocol whose communication complexity is asymptotically as small as a *semi-honest* garbled-circuit protocol in the OT-hybrid model.
- If the preprocessing phase is carried out by a trusted server, we obtain a constant-round protocol whose communication complexity is essentially the same as in the linear-round protocol of Mohassel et al. in the analogous setting.

1 Introduction

There have been incredible advances in the efficiency of protocols for (malicious) secure two-party computation (2PC) over the last decade. One popular approach for designing such protocols is to apply the "cut-and-choose" technique [LP07, SS11, LP11, HKE13, Lin13, Bra13, FJN14, AMPR14] to Yao's garbled-circuit protocol [Yao86] for constant-round (semi-honest) secure two-party computation. For statistical security $2^{-\rho}$, the best protocols using this paradigm require the parties to generate, send, and evaluate ρ garbled circuits, which is optimal for the cut-and-choose approach. Recently, Wang et al. [WMK16] showed a protocol based on this technique that can securely evaluate an AES circuit (in the single-execution setting with no preprocessing) in only 65 ms.

The above approach incurs significant overhead when large circuits are evaluated precisely because ρ garbled circuits need to be transmitted. In order to mitigate this drawback, recent works have explored secure computation in an *amortized* setting where the same function is evaluated multiple times (on different inputs) [HKK+14, LR14, LR15, RR16]. When amortizing over τ

Protocol	Circuit-independent preprocessing	Circuit-dependent preprocessing	Online phase	Storage
Cut-and-choose	—	$O\left(\lvert\mathcal{C}\rvert\rho\kappa\right)$	$O(\mathcal{I}\rvert\rho\kappa)$	$O(\lvert\mathcal{C}\rvert\rho\kappa)$
Amortized	—	$O\left(\lvert\mathcal{C}\rvert\frac{\rho}{\log\tau}\kappa\right)$	$O\left(\lvert\mathcal{I}\rvert\frac{\rho}{\log\tau}\kappa\right)$	$O\left(\frac{\lvert\mathcal{C}\rvert\rho\kappa}{\log\tau}\right)$
LEGO	$O\left(\frac{\lvert\mathcal{C}\rvert\rho\kappa}{\log\tau+\log\lvert\mathcal{C}\rvert}\right)$	$O(\lvert\mathcal{C}\rvert\kappa)$	$O(\lvert\mathcal{I}\rvert\kappa)$	$O\left(\frac{\lvert\mathcal{C}\rvert\rho\kappa}{\log\tau+\log\lvert\mathcal{C}\rvert}\right)$
SPDZ-BMR[1] [LPSY15]	$O(\lvert\mathcal{C}\rvert\kappa^2)$	$O(\lvert\mathcal{C}\rvert\kappa)$	$O(\lvert\mathcal{I}\rvert\kappa)$	$O(\lvert\mathcal{C}\rvert\kappa)$
This work with TinyOT	$O\left(\frac{\lvert\mathcal{C}\rvert\rho\kappa}{\log\tau+\log\lvert\mathcal{C}\rvert}\right)$	$O(\lvert\mathcal{C}\rvert\kappa)$	$\lvert\mathcal{I}\rvert\kappa$	$O(\lvert\mathcal{C}\rvert\kappa)$
This work with IPS[2]	$O(\lvert\mathcal{C}\rvert\kappa)$			

Table 1: **Communication complexity of constant-round 2PC protocols.** $|\mathcal{I}|$ represents the length of the inputs, and $|\mathcal{C}|$ denotes the circuit size. The statistical security parameter is ρ, and $\kappa \geq \rho$ denotes the computational security parameter. τ is the number of executions for protocols in the amortized setting. Storage is expressed as the amount of data to be stored after the preprocessing phase(s). Terms that are independent of the input/circuit size are ignored.

executions, only $O(\frac{\rho}{\log\tau})$ garbled circuits are needed per execution. Rindal and Rosulek [RR16] recently reported 6.4 ms for evaluation of an AES circuit, amortized over 1024 executions.

Other techniques for secure two-party computation, with asymptotically better performance than cut-and-choose (without amortization), have also been investigated. The LEGO protocol and subsequent optimizations [NO09, FJN+13, FJNT15, HZ15, NST17] are based on a gate-level cut-and-choose approach that can be carried out during a preprocessing phase before the circuit to be evaluated is known. This class of protocols has good asymptotic performance (see Table 1) and small online time; however, the overall cost of the state-of-the-art LEGO implementation [NST17] is still slightly higher than the overall cost of the best protocol based on the cut-and-choose approach applied at the garbled-circuit level.

The Beaver-Micali-Rogaway compiler [BMR90] provides yet another approach to constructing constant-round protocols secure against malicious adversaries. This compiler uses an "outer" secure-computation protocol to generate a garbled circuit. Lindell et al. [LPSY15] apply a similar idea using SPDZ [DPSZ12] as the outer protocol.

Protocols running in a super-constant number of rounds have also been investigated. The TinyOT protocol [NNOB12] adds malicious security to the classical GMW protocol [GMW87] by adding an information-theoretic MAC to the shares held by the parties. TinyOT has smaller communication complexity than the LEGO family of protocols, but—just like the GMW protocol—has round complexity linear in the depth of the circuit being evaluated. (In contrast, all the results cited previously run in a constant number of rounds.) The IPS compiler [IPS08, LOP11] has asymptotic communication complexity (in the OT-hybrid model) proportional to the size of the underlying circuit being evaluated. It, too, has the disadvantage of requiring a number of rounds linear in the depth of the circuit. A more serious drawback is that the concrete complexity of the protocol is unclear, since it has not yet been implemented (and appears quite difficult to implement).

[1] Based on [KOS16]; the complexity of circuit-independent preprocessing can be reduced to $O(|\mathcal{C}|\kappa)$ using somewhat homomorphic encryption [DPSZ12], but at the expense of requiring a number of public-key operations proportional to the circuit size.

[2] In the bit-OT-hybrid model.

In Table 1, we summarize the communication complexity of the various approaches for constructing constant-round 2PC protocols. Following [NST17], we divide execution of protocols into three phases:

- **Function-independent preprocessing.** During this phase, the parties need not know their inputs nor the function to be computed (beyond an upper bound on the number of gates).
- **Function-dependent preprocessing.** In this phase, the parties know what function they will compute, but need not know their inputs.

  Often, the first two phases are combined and referred to simply as the *offline phase*.
- **Online phase.** In this phase, two parties evaluate the agreed-upon function on their respective inputs.

Our contribution. We propose a new approach for constant-round 2PC protocols with extremely low communication complexity. At a high level (further details are given in Section 3), our protocol relies on a function-independent preprocessing phase to realize an ideal functionality that we call $\mathcal{F}_{\mathsf{Pre}}$. Following ideas of [NNOB12], we use this preprocessing phase to set up correlated information at the two parties that they can use during the online phase for information-theoretic authentication of different values. In contrast to [NNOB12], however, the parties in our protocol use this information in the online phase to distributively generate a *single* "authenticated" garbled circuit. (Conceptually similar ideas were used by Damgård and Ishai [DI05] in the context of multiparty computation with honest majority, and by Choi et al. [CKMZ14] for three-party computation with dishonest majority.) As in the semi-honest case, this garbled circuit can then be transmitted and evaluated using just one additional round of interaction.

Frederiksen et al. [FJNT15] introduces the notion of interactive garbling schemes and how to use it to construct a 2PC protocol. Our work also constructs garbled circuits interactively, but the underlying techniques and ideas are fundamentally different: in our protocol, garbled circuits are distributed to two parties and authenticated to the evaluator.

Regardless of how we realize $\mathcal{F}_{\mathsf{Pre}}$, our protocol is extremely efficient in the function-dependent preprocessing phase and the online phase. Specifically, compared to a *semi-honest* garbled-circuit protocol, the cost of the function-dependent preprocessing phase of our protocol is only about 30% higher (assuming 128-bit computational security and 40-bit statistical security), and the cost of the online phase is essentially unchanged. The cost of the function-independent preprocessing phase—and thus the cost of the entire protocol—depends on precisely how we realize $\mathcal{F}_{\mathsf{Pre}}$:

- If $\mathcal{F}_{\mathsf{Pre}}$ is instantiated using TinyOT, the asymptotic communication complexity of our protocol is as good as in protocols based on LEGO, but with two advantages. First, our protocol has better *concrete* communication complexity (see Section 8), especially in the online phase, and overall cost. Furthermore, the amount of storage needed by our protocol between the preprocessing phase and the online phase is (asymptotically) smaller. The latter is especially important when very large circuits are evaluated (see Table 1). We further improve the concrete efficiency by describing several improvements to TinyOT in Section 6.

  Compared to the protocol of Lindell et al. [LPSY15], our protocol is asymptotically more efficient in the function-independent preprocessing phase; more importantly, the concrete efficiency of our protocol is much better since our work is compatible with free-XOR and we do not suffer from any blowup in the size of the circuit being evaluated. In particular, Lindell et al. require five SPDZ-style multiplications per AND gate of the underlying circuit, while we only need one TinyOT-style AND computation per AND gate.

- When $\mathcal{F}_{\mathsf{Pre}}$ is instantiated using the IPS compiler, we obtain what is (to the best of our knowledge) the *first* constant-round protocol with communication complexity $O(|\mathcal{C}|\kappa)$ in the OT-hybrid model. Note that this (asymptotically) matches the communication complexity of *semi-honest* secure two-party computation based on garbled circuits.

- We can also realize $\mathcal{F}_{\mathsf{Pre}}$ using a (semi-)trusted server. In that case we obtain a constant-round protocol for server-aided 2PC with total communication $O(|\mathcal{C}|\kappa)$. Previous work in the same model [MOR16] achieves the same communication complexity but with number of rounds proportional to the circuit depth.

2 Notations and Preliminaries

We use κ to denote the computational security parameter (i.e., security should hold against attackers running in time $\approx 2^\kappa$), and ρ for the statistical security parameter (i.e., an adversary should succeed in cheating with probability at most $2^{-\rho}$). We use $=$ to denote equality and $:=$ to denote assignment.

A circuit is represented as a list of gates having the format $(\alpha, \beta, \gamma, T)$, where α and β denote the input-wire indices of the gate, γ denotes the output-wire index of the gate, and $T \in \{\oplus, \wedge\}$ denotes the type of the gate. We use $\mathcal{I}_1$ to denote the set of input-wire indices for $\mathsf{P_A}$'s input, $\mathcal{I}_2$ to denote the set of input-wire indices for $\mathsf{P_B}$'s input, $\mathcal{W}$ to denote the set of output-wire indices of all AND gates, and $\mathcal{O}$ to denote the set of output-wire indices of the circuit. We denote the parties running the secure-computation protocol by $\mathsf{P_A}$ and $\mathsf{P_B}$.

2.1 Information-theoretic MACs

We use the information-theoretic message authentication codes (IT-MAC) of [NNOB12]. $\mathsf{P_A}$ holds a random *global key* $\Delta_{\mathsf{A}} \in \{0,1\}^\rho$. A bit b known by $\mathsf{P_B}$ can be authenticated by having $\mathsf{P_A}$ hold a random key $\mathsf{K}[b]$ and having $\mathsf{P_B}$ hold the corresponding tag $\mathsf{M}[b] := \mathsf{K}[b] \oplus b\Delta_{\mathsf{A}}$. Symmetrically, $\mathsf{P_B}$ holds an independent global key Δ_{B}; a bit b known by $\mathsf{P_A}$ is authenticated by having $\mathsf{P_B}$ hold a random key $\mathsf{K}[b]$ and having $\mathsf{P_A}$ hold the tag $\mathsf{M}[b] := \mathsf{K}[b] \oplus b\Delta_{\mathsf{B}}$. We use $[b]_{\mathsf{A}}$ to denote an authenticated bit known to $\mathsf{P_A}$ (i.e., $[b]_{\mathsf{A}}$ means that $\mathsf{P_A}$ holds $(b, \mathsf{M}[b])$ and $\mathsf{P_B}$ holds $\mathsf{K}[b]$), and $[b]_{\mathsf{B}}$ is defined symmetrically.

Observe that this MAC is XOR-homomorphic: given $[b]_{\mathsf{A}}$ and $[c]_{\mathsf{A}}$, the parties can (locally) compute $[b \oplus c]_{\mathsf{A}}$ by having $\mathsf{P_A}$ compute $\mathsf{M}[b \oplus c] = \mathsf{M}[b] \oplus \mathsf{M}[c]$ and $\mathsf{P_B}$ compute $\mathsf{K}[b \oplus c] := (\mathsf{K}[b] \oplus \mathsf{K}[c])$.

It is possible to extend the above idea to XOR-shared values by having each party's share be authenticated. That is, say we have a value $\lambda := r \oplus s$, where $\mathsf{P_A}$ knows r and $\mathsf{P_B}$ knows s. Then by having $\mathsf{P_A}$ hold $(r, \mathsf{M}[r], \mathsf{K}[s])$ and $\mathsf{P_B}$ hold $(s, \mathsf{K}[r], \mathsf{M}[s])$, we end up with an authenticated secret-sharing of λ. It can be observed that this scheme is also XOR-homomorphic.

As described in the Introduction, we use a preprocessing phase that realizes a stateful ideal functionality $\mathcal{F}_{\mathsf{Pre}}$. This functionality, described in Figure 1, is used to set up correlated values between the players along with their corresponding IT-MACs. The functionality chooses uniform global keys (once-and-for-all) for each party, with the malicious party being allowed to choose its global key. Then, when the parties request a random authenticated bit, the functionality generates

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

- Upon receiving Δ_{A} from $\mathsf{P_A}$ and init from $\mathsf{P_B}$, and assuming no values $\Delta_{\mathsf{A}}, \Delta_{\mathsf{B}}$ are currently stored, choose uniform $\Delta_{\mathsf{B}} \in \{0,1\}^\rho$ and store $\Delta_{\mathsf{A}}, \Delta_{\mathsf{B}}$. Send Δ_{B} to $\mathsf{P_B}$.
- Upon receiving $(\mathsf{random}, r, \mathsf{M}[r], \mathsf{K}[s])$ from $\mathsf{P_A}$ and random from $\mathsf{P_B}$, sample uniform $s \in \{0,1\}$ and set $\mathsf{K}[r] := \mathsf{M}[r] \oplus r\Delta_{\mathsf{B}}$ and $\mathsf{M}[s] := \mathsf{K}[s] \oplus s\Delta_{\mathsf{A}}$. Send $(s, \mathsf{M}[s], \mathsf{K}[r])$ to $\mathsf{P_B}$.
- Upon receiving $(\mathtt{AND}, (r_1, \mathsf{M}[r_1], \mathsf{K}[s_1]), (r_2, \mathsf{M}[r_2], \mathsf{K}[s_2]), r_3, \mathsf{M}[r_3], \mathsf{K}[s_3])$ from $\mathsf{P_A}$, and $(\mathtt{AND}, (s_1, \mathsf{M}[s_1], \mathsf{K}[r_1]), (s_2, \mathsf{M}[s_2], \mathsf{K}[r_2]))$ from $\mathsf{P_B}$, verify that $\mathsf{M}[r_i] = \mathsf{K}[r_i] \oplus r_i\Delta_{\mathsf{B}}$ and that $\mathsf{M}[s_i] = \mathsf{K}[s_i] \oplus s_i\Delta_{\mathsf{A}}$ for $i \in \{1,2\}$ and send cheat to $\mathsf{P_B}$ if not. Otherwise, set $s_3 := r_3 \oplus ((r_1 \oplus s_1) \wedge (r_2 \oplus s_2))$ and set $\mathsf{K}[r_3] := \mathsf{M}[r_3] \oplus r_3\Delta_{\mathsf{B}}$ and $\mathsf{M}[s_3] := \mathsf{K}[s_3] \oplus s_3\Delta_{\mathsf{A}}$. Send $(s_3, \mathsf{M}[s_3], \mathsf{K}[r_3])$ to $\mathsf{P_B}$.

Figure 1: The preprocessing functionality, assuming $\mathsf{P_A}$ is corrupted. (It is defined symmetrically if $\mathsf{P_B}$ is corrupted. If neither party is corrupted, the functionality is adapted in the obvious way.)

an authenticated secret sharing of the random bit $r \oplus s$. (The malicious party may choose the "random values" it receives, but note that this does not reveal anything about $r \oplus s$ or the other party's global key to the adversary.) Finally, the parties may also submit their authenticated shares for two bits; the functionality then computes a (fresh) authenticated share of the AND of those bits. We defer until Section 4.2 a discussion of how $\mathcal{F}_{\mathsf{Pre}}$ can be instantiated.

3 Protocol Intuition

We give a high-level overview of the core of our protocol in the $\mathcal{F}_{\mathsf{Pre}}$-hybrid model. Our protocol is based on a garbled circuit that the parties compute in a distributed fashion, where the garbled circuit is "authenticated" in the sense that the circuit generator ($\mathsf{P_A}$ in our case) cannot change the logic of the circuit. We describe the intuition behind the garbled circuit we use in several steps.

We begin by reviewing standard garbled circuits. Each wire α of a circuit is associated with a random "mask" $\lambda_\alpha \in \{0,1\}$ known to $\mathsf{P_A}$. If the true value (i.e., the value when the circuit is evaluated on the parties' inputs) of that wire is x, then the masked value observed by the circuit evaluator (namely, $\mathsf{P_B}$) on that wire will be $\bar{x} = x \oplus \lambda_\alpha$. Each wire α is also associated with two labels $\mathsf{L}_{\alpha,0}$ and $\mathsf{L}_{\alpha,1} := \mathsf{L}_{\alpha,0} \oplus \Delta$ known to $\mathsf{P_A}$ (here we are using the free-XOR technique[KS08]). If the masked bit on that wire is $\bar{x}$, then $\mathsf{P_B}$ learns $\mathsf{L}_{\alpha,\bar{x}}$.

Let H be a hash function modeled as a random oracle. The garbled table for, e.g., an and-gate $(\alpha, \beta, \gamma, \wedge)$ is given by:

$\bar{x} = x \oplus \lambda_\alpha$	$\bar{y} = y \oplus \lambda_\beta$	Truth Table	Garbled Table
0	0	$\bar{z}_{00} = (\lambda_\alpha \wedge \lambda_\beta) \oplus \lambda_\gamma$	$H(\mathsf{L}_{\alpha,0}, \mathsf{L}_{\beta,0}, \gamma, 00) \oplus (\bar{z}_{00}, L_{\gamma,\bar{z}_{00}})$
0	1	$\bar{z}_{01} = (\lambda_\alpha \wedge \bar{\lambda}_\beta) \oplus \lambda_\gamma$	$H(\mathsf{L}_{\alpha,0}, \mathsf{L}_{\beta,1}, \gamma, 01) \oplus (\bar{z}_{01}, L_{\gamma,\bar{z}_{01}})$
1	0	$\bar{z}_{10} = (\bar{\lambda}_\alpha \wedge \lambda_\beta) \oplus \lambda_\gamma$	$H(\mathsf{L}_{\alpha,1}, \mathsf{L}_{\beta,0}, \gamma, 10) \oplus (\bar{z}_{10}, L_{\gamma,\bar{z}_{10}})$
1	1	$\bar{z}_{11} = (\bar{\lambda}_\alpha \wedge \bar{\lambda}_\beta) \oplus \lambda_\gamma$	$H(\mathsf{L}_{\alpha,1}, \mathsf{L}_{\beta,1}, \gamma, 11) \oplus (\bar{z}_{11}, L_{\gamma,\bar{z}_{11}})$

$\mathsf{P_B}$, holding $(\bar{x}, L_{\alpha,\bar{x}})$ and $(\bar{y}, L_{\beta,\bar{y}})$, evaluates this garbled gate by picking the $(\bar{x}, \bar{y})$-row and decrypting using the garbled labels it holds, thus obtaining $(\bar{z}, L_{\gamma,\bar{z}})$.

The standard garbled circuit just described ensures security against a malicious $\mathsf{P_B}$, since (in an intuitive sense) $\mathsf{P_B}$ learns no information about the true values on any of the wires. Unfortunately, it provides no security against a malicious $\mathsf{P_A}$ who can potentially cheat by corrupting rows in the various garbled tables. One particular attack a malicious $\mathsf{P_A}$ can carry out is a *selective-failure* attack. Say, for example, that a malicious $\mathsf{P_A}$ corrupts only the $(0,0)$-row of the garbled table for the gate above, and assume $\mathsf{P_B}$ aborts if it detects an error during evaluation. If $\mathsf{P_B}$ aborts, then $\mathsf{P_A}$ learns that the masked values on the input wires of the gate above were $\bar{x} = \bar{y} = 0$, from which it learns that the true values on those wires were λ_α and λ_β.

The selective-failure attack just mentioned can be prevented if the masks are hidden from $\mathsf{P_A}$. (In that case even if $\mathsf{P_A}$ learns the masked wire values as before, it learns nothing about the true wire values.) Since knowledge of the garbled table would leak information about the masks to $\mathsf{P_A}$, the garbled table must be hidden from $\mathsf{P_A}$ as well. That is, we now want to set up a situation in which $\mathsf{P_A}$ and $\mathsf{P_B}$ hold *secret shares* of the garbled table, as follows:

$\bar{x} = x \oplus \lambda_\alpha$	$\bar{y} = y \oplus \lambda_\beta$	Truth Table	$\mathsf{P_A}$'s share of Garbled Table	$\mathsf{P_B}$'s share of Garbled Table
0	0	$\bar{z}_{00} = (\lambda_\alpha \wedge \lambda_\beta) \oplus \lambda_\gamma$	$H(\mathsf{L}_{\alpha,0}, \mathsf{L}_{\beta,0}, \gamma, 00) \oplus (r_{00}, R_{00} \oplus L_{\gamma,\bar{z}_{00}})$	$(s_{00} = \bar{z}_{00} \oplus r_{00}, R_{00})$
0	1	$\bar{z}_{01} = (\lambda_\alpha \wedge \bar{\lambda}_\beta) \oplus \lambda_\gamma$	$H(\mathsf{L}_{\alpha,0}, \mathsf{L}_{\beta,1}, \gamma, 01) \oplus (r_{01}, R_{01} \oplus L_{\gamma,\bar{z}_{01}})$	$(s_{01} = \bar{z}_{01} \oplus r_{01}, R_{01})$
1	0	$\bar{z}_{10} = (\bar{\lambda}_\alpha \wedge \lambda_\beta) \oplus \lambda_\gamma$	$H(\mathsf{L}_{\alpha,1}, \mathsf{L}_{\beta,0}, \gamma, 10) \oplus (r_{10}, R_{10} \oplus L_{\gamma,\bar{z}_{10}})$	$(s_{10} = \bar{z}_{10} \oplus r_{10}, R_{10})$
1	1	$\bar{z}_{11} = (\bar{\lambda}_\alpha \wedge \bar{\lambda}_\beta) \oplus \lambda_\gamma$	$H(\mathsf{L}_{\alpha,1}, \mathsf{L}_{\beta,1}, \gamma, 11) \oplus (r_{11}, R_{11} \oplus L_{\gamma,\bar{z}_{11}})$	$(s_{11} = \bar{z}_{11} \oplus r_{11}, R_{11})$

Once $\mathsf{P_A}$ sends its shares of all the garbled gates, $\mathsf{P_B}$ can evaluate the garbled circuit: Given $(\bar{x}, L_{\alpha,\bar{x}})$ and $(\bar{y}, L_{\beta,\bar{y}})$, it picks the appropriate row, decrypts $\mathsf{P_A}$'s share of that row using the garbed labels it holds, and then XORs the result with its own shares of that same row to obtain $(\bar{z}, L_{\gamma,\bar{z}})$.

Informally, the above modification ensures *privacy* against a malicious $\mathsf{P_A}$ since (intuitively) the result of any changes $\mathsf{P_A}$ introduces will depend on the random masks but be *independent* of $\mathsf{P_B}$'s inputs. However, $\mathsf{P_A}$ can still affect *correctness* by, e.g., flipping the masked value in one of the rows of a garbled gate. This can be addressed by adding an information-theoretic MAC on $\mathsf{P_A}$'s share of the masked bit. That is, the shares of the garbled table now take the following form:

$\bar{x} = x \oplus \lambda_\alpha$	$\bar{y} = y \oplus \lambda_\beta$	$\mathsf{P_A}$'s share of Garbled Table	$\mathsf{P_B}$'s share of Garbled Table
0	0	$H(\mathsf{L}_{\alpha,0}, \mathsf{L}_{\beta,0}, \gamma, 00) \oplus (r_{00}, \mathsf{M}[r_{00}], R_{00} \oplus L_{\gamma,\bar{z}_{00}})$	$(s_{00} = \bar{z}_{00} \oplus r_{00}, \mathsf{K}[r_{00}], R_{00})$
0	1	$H(\mathsf{L}_{\alpha,0}, \mathsf{L}_{\beta,1}, \gamma, 01) \oplus (r_{01}, \mathsf{M}[r_{01}], R_{01} \oplus L_{\gamma,\bar{z}_{01}})$	$(s_{01} = \bar{z}_{01} \oplus r_{01}, \mathsf{K}[r_{01}], R_{01})$
1	0	$H(\mathsf{L}_{\alpha,1}, \mathsf{L}_{\beta,0}, \gamma, 10) \oplus (r_{10}, \mathsf{M}[r_{10}], R_{10} \oplus L_{\gamma,\bar{z}_{10}})$	$(s_{10} = \bar{z}_{10} \oplus r_{10}, \mathsf{K}[r_{10}], R_{10})$
1	1	$H(\mathsf{L}_{\alpha,1}, \mathsf{L}_{\beta,1}, \gamma, 11) \oplus (r_{11}, \mathsf{M}[r_{11}], R_{11} \oplus L_{\gamma,\bar{z}_{11}})$	$(s_{11} = \bar{z}_{11} \oplus r_{11}, \mathsf{K}[r_{11}], R_{11})$

Once $\mathsf{P_A}$ sends its shares of the garbled circuit to $\mathsf{P_B}$, the garbled circuit can be evaluated as before. Now, however, $\mathsf{P_B}$ will verify the MAC on $\mathsf{P_A}$'s share of each masked bit that it learns. This limits $\mathsf{P_A}$ to only being able to cause $\mathsf{P_B}$ to abort; as before, though, any such abort will occur independent of $\mathsf{P_B}$'s actual input.

Efficient realization. Although the above idea is powerful, it still remains to design an efficient protocol that allows the parties to distributively compute shares of a garbled table of the above form even when one of the parties is malicious. One key observation is that $\mathsf{P_A}$'s shares of the wire labels need not be authenticated; in the worst-case, incorrect values used by $\mathsf{P_A}$ will cause an input-*independent* abort. Note further that, for example,

$$\begin{aligned}
\mathsf{L}_{\gamma,\bar{z}_{00}} &= \mathsf{L}_{\gamma,0} \oplus \bar{z}_{00}\Delta_\mathsf{A} \\
&= \mathsf{L}_{\gamma,0} \oplus (r_{00} \oplus s_{00})\Delta_\mathsf{A} \\
&= \mathsf{L}_{\gamma,0} \oplus r_{00}\Delta_\mathsf{A} \oplus s_{00}\Delta_\mathsf{A} \\
&= (\mathsf{L}_{\gamma,0} \oplus r_{00}\Delta_\mathsf{A} \oplus \mathsf{K}[s_{00}]) \oplus (\mathsf{K}[s_{00}] \oplus s_{00}\Delta_\mathsf{A}).
\end{aligned}$$

Our next key observation is that if s_{00} is an authenticated bit known to $\mathsf{P_B}$, then $\mathsf{P_A}$ can locally compute $\mathsf{L}_{\gamma,0} \oplus r_{00}\Delta_\mathsf{A} \oplus \mathsf{K}[s_{00}]$; then the other share $\mathsf{K}[s_{00}] \oplus s_{00}\Delta_\mathsf{A}$ is just the MAC on s_{00} that $\mathsf{P_B}$ already knows! Thus, we can rewrite the garbled table as follows:

$x \oplus \lambda_\alpha$	$y \oplus \lambda_\beta$	$\mathsf{P_A}$'s share of Garbled Table	$\mathsf{P_B}$'s share of Garbled Table
0	0	$H(\mathsf{L}_{\alpha,0}, \mathsf{L}_{\beta,0}, \gamma, 00) \oplus (r_{00}, \mathsf{M}[r_{00}], L_{\gamma,0} \oplus r_{00}\Delta_\mathsf{A} \oplus \mathsf{K}[s_{00}])$	$(s_{00} = \bar{z}_{00} \oplus r_{00}, \mathsf{K}[r_{00}], \mathsf{M}[s_{00}])$
0	1	$H(\mathsf{L}_{\alpha,0}, \mathsf{L}_{\beta,1}, \gamma, 01) \oplus (r_{01}, \mathsf{M}[r_{01}], L_{\gamma,0} \oplus r_{01}\Delta_\mathsf{A} \oplus \mathsf{K}[s_{01}])$	$(s_{01} = \bar{z}_{01} \oplus r_{01}, \mathsf{K}[r_{01}], \mathsf{M}[s_{01}])$
1	0	$H(\mathsf{L}_{\alpha,1}, \mathsf{L}_{\beta,0}, \gamma, 10) \oplus (r_{10}, \mathsf{M}[r_{10}], L_{\gamma,0} \oplus r_{10}\Delta_\mathsf{A} \oplus \mathsf{K}[s_{10}])$	$(s_{10} = \bar{z}_{10} \oplus r_{10}, \mathsf{K}[r_{10}], \mathsf{M}[s_{10}])$
1	1	$H(\mathsf{L}_{\alpha,1}, \mathsf{L}_{\beta,1}, \gamma, 11) \oplus (r_{11}, \mathsf{M}[r_{11}], L_{\gamma,0} \oplus r_{11}\Delta_\mathsf{A} \oplus \mathsf{K}[s_{11}])$	$(s_{11} = \bar{z}_{11} \oplus r_{11}, \mathsf{K}[r_{11}], \mathsf{M}[s_{11}])$

(The $\{R_{ij}\}$ values are no longer needed since the $\{s_{ij}\}$ are unknown to $\mathsf{P_A}$, and that is enough to hide the masks from $\mathsf{P_A}$.) Shares of the table then become easy to compute in a distributed fashion.

Final optimization. One final optimization is based on the simple observation that the entries in the truth table are linearly dependent. More precisely,

$$\begin{aligned}
\bar{z}_{00} &= (\lambda_\alpha \wedge \lambda_\beta) \oplus \lambda_\gamma \\
\bar{z}_{01} &= (\lambda_\alpha \wedge \bar{\lambda}_\beta) \oplus \lambda_\gamma = \bar{z}_{00} \oplus \lambda_\alpha \\
\bar{z}_{10} &= (\bar{\lambda}_\alpha \wedge \lambda_\beta) \oplus \lambda_\gamma = \bar{z}_{00} \oplus \lambda_\beta \\
\bar{z}_{11} &= (\bar{\lambda}_\alpha \wedge \bar{\lambda}_\beta) \oplus \lambda_\gamma = \bar{z}_{01} \oplus \lambda_\beta \oplus 1
\end{aligned}$$

Therefore, in order to jointly compute the above table, the parties just need to get MACs on shares of the masks $\lambda_\alpha, \lambda_\beta, \lambda_\gamma$, and then compute the MACs on shares of the bit $\lambda_\alpha \wedge \lambda_\beta$.

4 Framework for Our Protocols

4.1 Protocol in the $\mathcal{F}_{\mathsf{Pre}}$-Hybrid Model

In Figure 2, we give the complete description of our main protocol in the $\mathcal{F}_{\mathsf{Pre}}$-hybrid model. Note that the calls to $\mathcal{F}_{\mathsf{Pre}}$ can be performed in parallel, so the protocol runs in constant rounds. Since we show below that $\mathcal{F}_{\mathsf{Pre}}$ can be realized efficiently by constant-round protocols, this gives a protocol (in the plain model) with overall constant round complexity.

Although our protocol calls $\mathcal{F}_{\mathsf{Pre}}$ in the function-dependent preprocessing phase, it is easy to push this to the function-independent phase using standard techniques. The protocol can be easily extended to support reactive computation. We leave it as future work to figure out further details.

4.2 Instantiation of $\mathcal{F}_{\mathsf{Pre}}$

In the following, we discuss various ways $\mathcal{F}_{\mathsf{Pre}}$ can be instantiated.

TinyOT-based instantiation. We can instantiate $\mathcal{F}_{\mathsf{Pre}}$ using TinyOT. (We describe some improvements to the TinyOT protocol in Section 6.) In fact, rather than using TinyOT itself to realize the $\mathcal{F}_{\mathsf{Pre}}$ functionality, we can instantiate $\mathcal{F}_{\mathsf{Pre}}$ directly based on the $\mathcal{F}_{\mathsf{DEAL}}$ functionality defined in the TinyOT paper. One technical issue is that the $\mathcal{F}_{\mathsf{DEAL}}$ functionality defined there includes a "global key query" for technical reasons. This can be added to our $\mathcal{F}_{\mathsf{Pre}}$ functionality without affecting the proof much. We will provide further details in the full version.

IPS-based instantiation. We can use the IPS protocol to realize the $\mathcal{F}_{\mathsf{Pre}}$ functionality. In the function-dependent preprocessing phase, we need to produce a sharing of λ_i for each wire i,

Protocol $\Pi_{\mathbf{2pc}}$

Inputs: In the function-dependent phase, the parties agree on a circuit for a function $f : \{0,1\}^{|\mathcal{I}_1|} \times \{0,1\}^{|\mathcal{I}_2|} \to \{0,1\}^{|\mathcal{O}|}$. In the input-processing phase, $\mathsf{P_A}$ holds $x \in \{0,1\}^{|\mathcal{I}_1|}$ and $\mathsf{P_A}$ holds $y \in \{0,1\}^{|\mathcal{I}_2|}$.

Function-independent preprocessing:

1. $\mathsf{P_A}$ and $\mathsf{P_B}$ send init to $\mathcal{F}_{\mathsf{Pre}}$, which sends Δ_{A} to $\mathsf{P_A}$ and Δ_{B} to $\mathsf{P_B}$.
2. For each wire $w \in \mathcal{I}_1 \cup \mathcal{I}_2 \cup \mathcal{W}$, parties $\mathsf{P_A}$ and $\mathsf{P_B}$ send random to $\mathcal{F}_{\mathsf{Pre}}$. In return, $\mathcal{F}_{\mathsf{Pre}}$ sends $(r_w, \mathsf{M}[r_w], \mathsf{K}[s_w])$ to $\mathsf{P_A}$ and $(s_w, , \mathsf{M}[s_w], \mathsf{K}[r_w])$ to $\mathsf{P_B}$, where $\lambda_w = s_w \oplus r_w$. $\mathsf{P_A}$ also picks a uniform κ-bit string $\mathsf{L}_{w,0}$.

Function-dependent preprocessing:

3. For each gate $\mathcal{G} = (\alpha, \beta, \gamma, \oplus)$, $\mathsf{P_A}$ computes $(r_\gamma, \mathsf{M}[r_\gamma], \mathsf{K}[s_\gamma]) := (r_\alpha \oplus r_\beta, \mathsf{M}[r_\alpha] \oplus \mathsf{M}[r_\beta], \mathsf{K}[s_\alpha] \oplus \mathsf{K}[s_\beta])$ and $\mathsf{L}_{\gamma,0} := \mathsf{L}_{\alpha,0} \oplus \mathsf{L}_{\beta,0}$. $\mathsf{P_B}$ computes $(s_\gamma, \mathsf{M}[s_\gamma], \mathsf{K}[r_\gamma]) := (s_\alpha \oplus s_\beta, \mathsf{M}[r_\beta] \oplus \mathsf{M}[r_\beta], \mathsf{K}[r_\alpha] \oplus \mathsf{K}[r_\beta])$.
4. Then, for each gate $\mathcal{G} = (\alpha, \beta, \gamma, \wedge)$:

   (a) $\mathsf{P_A}$ (resp., $\mathsf{P_B}$) sends $(\mathtt{and}, (r_\alpha, \mathsf{M}[r_\alpha], \mathsf{K}[s_\alpha]), (r_\beta, \mathsf{M}[r_\beta], \mathsf{K}[s_\beta]))$ (resp., $(\mathtt{and}, (s_\alpha, \mathsf{M}[s_\alpha], \mathsf{K}[r_\alpha]), (s_\beta, \mathsf{M}[s_\beta], \mathsf{K}[r_\beta])))$ to $\mathcal{F}_{\mathsf{Pre}}$. In return, $\mathcal{F}_{\mathsf{Pre}}$ sends $(r_\sigma, \mathsf{M}[r_\sigma], \mathsf{K}[s_\sigma])$ to $\mathsf{P_A}$ and $(s_\sigma, \mathsf{M}[s_\sigma], \mathsf{K}[r_\sigma])$ to $\mathsf{P_B}$, where $s_\sigma \oplus r_\sigma = \lambda_\alpha \wedge \lambda_\beta$.

   (b) $\mathsf{P_A}$ computes the following locally:

$$
\begin{aligned}
(r_{\gamma,0}, \mathsf{M}[r_{\gamma,0}], \mathsf{K}[s_{\gamma,0}]) &:= (r_\sigma \oplus r_\gamma, && \mathsf{M}[r_\sigma] \oplus \mathsf{M}[r_\gamma], && \mathsf{K}[s_\sigma] \oplus \mathsf{K}[s_\gamma] &&)\\
(r_{\gamma,1}, \mathsf{M}[r_{\gamma,1}], \mathsf{K}[s_{\gamma,1}]) &:= (r_\sigma \oplus r_\gamma \oplus r_\alpha, && \mathsf{M}[r_\sigma] \oplus \mathsf{M}[r_\gamma] \oplus \mathsf{M}[r_\alpha], && \mathsf{K}[s_\sigma] \oplus \mathsf{K}[s_\gamma] \oplus \mathsf{K}[s_\alpha] &&)\\
(r_{\gamma,2}, \mathsf{M}[r_{\gamma,2}], \mathsf{K}[s_{\gamma,2}]) &:= (r_\sigma \oplus r_\gamma \oplus r_\beta, && \mathsf{M}[r_\sigma] \oplus \mathsf{M}[r_\gamma] \oplus \mathsf{M}[r_\beta], && \mathsf{K}[s_\sigma] \oplus \mathsf{K}[s_\gamma] \oplus \mathsf{K}[s_\beta] &&)\\
(r_{\gamma,3}, \mathsf{M}[r_{\gamma,3}], \mathsf{K}[s_{\gamma,3}]) &:= (r_\sigma \oplus r_\gamma \oplus r_\alpha \oplus r_\beta, && \mathsf{M}[r_\sigma] \oplus \mathsf{M}[r_\gamma] \oplus \mathsf{M}[r_\alpha] \oplus \mathsf{M}[r_\beta], && \mathsf{K}[s_\sigma] \oplus \mathsf{K}[s_\gamma] \oplus \mathsf{K}[s_\alpha] \oplus \mathsf{K}[s_\beta] \oplus \Delta_{\mathsf{A}} &&)
\end{aligned}
$$

   (c) $\mathsf{P_B}$ computes the following locally:

$$
\begin{aligned}
(s_{\gamma,0}, \mathsf{M}[s_{\gamma,0}], \mathsf{K}[r_{\gamma,0}]) &:= (s_\sigma \oplus s_\gamma, && \mathsf{M}[s_\sigma] \oplus \mathsf{M}[s_\gamma], && \mathsf{K}[r_\sigma] \oplus \mathsf{K}[r_\gamma] &&)\\
(s_{\gamma,1}, \mathsf{M}[s_{\gamma,1}], \mathsf{K}[r_{\gamma,1}]) &:= (s_\sigma \oplus s_\gamma \oplus s_\alpha, && \mathsf{M}[s_\sigma] \oplus \mathsf{M}[s_\gamma] \oplus \mathsf{M}[s_\alpha], && \mathsf{K}[r_\sigma] \oplus \mathsf{K}[r_\gamma] \oplus \mathsf{K}[r_\alpha] &&)\\
(s_{\gamma,2}, \mathsf{M}[s_{\gamma,2}], \mathsf{K}[r_{\gamma,2}]) &:= (s_\sigma \oplus s_\gamma \oplus s_\beta, && \mathsf{M}[s_\sigma] \oplus \mathsf{M}[s_\gamma] \oplus \mathsf{M}[s_\beta], && \mathsf{K}[r_\sigma] \oplus \mathsf{K}[r_\gamma] \oplus \mathsf{K}[r_\beta] &&)\\
(s_{\gamma,3}, \mathsf{M}[s_{\gamma,3}], \mathsf{K}[r_{\gamma,3}]) &:= (s_\sigma \oplus s_\gamma \oplus s_\alpha \oplus s_\beta \oplus 1, && \mathsf{M}[s_\sigma] \oplus \mathsf{M}[s_\gamma] \oplus \mathsf{M}[s_\alpha] \oplus \mathsf{M}[s_\beta], && \mathsf{K}[r_\sigma] \oplus \mathsf{K}[r_\gamma] \oplus \mathsf{K}[r_\alpha] \oplus \mathsf{K}[r_\beta] &&)
\end{aligned}
$$

   (d) $\mathsf{P_A}$ computes $\mathsf{L}_{\alpha,1} := \mathsf{L}_{\alpha,0} \oplus \Delta_{\mathsf{A}}$ and $\mathsf{L}_{\beta,1} := \mathsf{L}_{\beta,0} \oplus \Delta_{\mathsf{A}}$, and then sends the following to $\mathsf{P_B}$:

$$
\begin{aligned}
G_{\gamma,0} &:= H(\mathsf{L}_{\alpha,0}, \mathsf{L}_{\beta,0}, \gamma, 0) \oplus (r_{\gamma,0}, \mathsf{M}[r_{\gamma,0}], \mathsf{L}_{\gamma,0} \oplus \mathsf{K}[s_{\gamma,0}] \oplus r_{\gamma,0}\Delta_{\mathsf{A}})\\
G_{\gamma,1} &:= H(\mathsf{L}_{\alpha,0}, \mathsf{L}_{\beta,1}, \gamma, 1) \oplus (r_{\gamma,1}, \mathsf{M}[r_{\gamma,1}], \mathsf{L}_{\gamma,0} \oplus \mathsf{K}[s_{\gamma,1}] \oplus r_{\gamma,1}\Delta_{\mathsf{A}})\\
G_{\gamma,2} &:= H(\mathsf{L}_{\alpha,1}, \mathsf{L}_{\beta,0}, \gamma, 2) \oplus (r_{\gamma,2}, \mathsf{M}[r_{\gamma,2}], \mathsf{L}_{\gamma,0} \oplus \mathsf{K}[s_{\gamma,2}] \oplus r_{\gamma,2}\Delta_{\mathsf{A}})\\
G_{\gamma,3} &:= H(\mathsf{L}_{\alpha,1}, \mathsf{L}_{\beta,1}, \gamma, 3) \oplus (r_{\gamma,3}, \mathsf{M}[r_{\gamma,3}], \mathsf{L}_{\gamma,0} \oplus \mathsf{K}[s_{\gamma,3}] \oplus r_{\gamma,3}\Delta_{\mathsf{A}})
\end{aligned}
$$

Input processing:

5. For each $w \in \mathcal{I}_1$, $\mathsf{P_A}$ sends $(r_w, \mathsf{M}[r_w])$ to $\mathsf{P_B}$, who checks that $(r_w, \mathsf{M}[r_w], \mathsf{K}[r_w])$ is valid. $\mathsf{P_B}$ then sends $y_w \oplus \lambda_w := s_w \oplus y_w \oplus r_w$ to $\mathsf{P_A}$. Finally, $\mathsf{P_A}$ sends $\mathsf{L}_{w, y_w \oplus \lambda_w}$ to $\mathsf{P_B}$.
6. For each $w \in \mathcal{I}_2$, $\mathsf{P_B}$ sends $(s_w, \mathsf{M}[s_w])$ to $\mathsf{P_A}$, who checks that $(s_w, \mathsf{M}[s_w], \mathsf{K}[s_w])$ is valid. $\mathsf{P_A}$ then sends $x_w \oplus \lambda_w := s_w \oplus x_w \oplus r_w$ and $\mathsf{L}_{w, x_w \oplus \lambda_w}$ to $\mathsf{P_B}$.

Circuit evaluation:

7. $\mathsf{P_B}$ evaluates the circuit in topological order. For each gate $\mathcal{G} = (\alpha, \beta, \gamma, T)$, $\mathsf{P_B}$ initially holds $(z_\alpha \oplus \lambda_\alpha, \mathsf{L}_{\alpha, z_\alpha \oplus \lambda_\alpha})$ and $(z_\beta \oplus \lambda_\beta, \mathsf{L}_{\beta, z_\beta \oplus \lambda_\beta})$, where z_α, z_β are the underlying values of the wires.

   (a) If $T = \oplus$, $\mathsf{P_B}$ computes $z_\gamma \oplus \lambda_\gamma := (z_\alpha \oplus \lambda_\alpha) \oplus (z_\beta \oplus \lambda_\beta)$ and $\mathsf{L}_{\gamma, z_\gamma \oplus \lambda_\gamma} := \mathsf{L}_{\alpha, z_\alpha \oplus \lambda_\alpha} \oplus \mathsf{L}_{\beta, z_\beta \oplus \lambda_\beta}$.

   (b) If $T = \wedge$, $\mathsf{P_B}$ computes $i := 2(z_\alpha \oplus \lambda_\alpha) + (z_\beta \oplus \lambda_\beta)$ followed by $(r_{\gamma,i}, \mathsf{M}[r_{\gamma,i}], \mathsf{L}_{\gamma,0} \oplus \mathsf{K}[s_{\gamma,i}] \oplus r_{\gamma,i}\Delta_{\mathsf{A}}) := G_{\gamma,i} \oplus H(\mathsf{L}_{\alpha, z_\alpha \oplus \lambda_\alpha}, \mathsf{L}_{\beta, z_\beta \oplus \lambda_\beta}, \gamma, i)$. Then $\mathsf{P_B}$ checks that $(r_{\gamma,i}, \mathsf{M}[r_{\gamma,i}], \mathsf{K}[r_{\gamma,i}])$ is valid and, if so, computes $z_\gamma \oplus \lambda_\gamma := (s_{\gamma,i} \oplus r_{\gamma,i})$ and $\mathsf{L}_{\gamma, z_\gamma \oplus \lambda_\gamma} := (\mathsf{L}_{\gamma,0} \oplus \mathsf{K}[s_{\gamma,i}] \oplus r_{\gamma,i}\Delta_{\mathsf{A}}) \oplus \mathsf{M}[s_{\gamma,i}]$.

Output determination:

8. For each $w \in \mathcal{O}$, $\mathsf{P_A}$ sends $(r_w, \mathsf{M}[r_w])$ to $\mathsf{P_B}$, who checks that $(r_w, \mathsf{M}[r_w], \mathsf{K}[r_w])$ is valid. If so, $\mathsf{P_B}$ computes $z_w := (\lambda_w \oplus z_w) \oplus r_w \oplus s_w$.

Figure 2: Our main protocol in the $\mathcal{F}_{\mathsf{Pre}}$-hybrid model.

and a sharing of $\lambda_\sigma = (\lambda_\alpha \wedge \lambda_\beta) \oplus \lambda_\gamma$ for each and-gate $(\alpha, \beta, \gamma, \wedge)$. These can be realized by a constant-depth circuit with $O((\kappa + \rho) \cdot |C|)$ gates. To evaluate a circuit of depth d and size

ℓ, the IPS protocol has communication complexity $O(\ell) + poly(\kappa, d, \log \ell)$ and $O(d)$ rounds of communication. When applied to our setting, this translates to a communication complexity of $O((\kappa+\rho) \cdot |C|) + poly(\kappa, \log |C|)$; for sufficiently large circuit size, the leading term is $O((\kappa+\rho) \cdot |C|)$. By using the IPS protocol we thus gain asymptotic improvements in communication complexity in the OT-hybrid model.

Using a trusted server. It is straightforward to instantiate $\mathcal{F}_{\mathsf{Pre}}$ using a trusted server. By applying the technique of Mohassel et al. [MOR16], the offline phase can also be decoupled from the identity of other party; we refer to their paper for further details.

5 Proof

5.1 Proof Intuition

The intuition of the protocol in Section 3 also provides some intuition of the security of the protocol. Here we provide more intuition on some key issues.

Selective-failure type attack. In most garbled circuit protocols, selective failure attacks can be launched on the input (by corrupting some garbled keys sent to oblivious transfer), as well as some internal wires (by corrupting some garbled rows in the garbled table). Input selective failure attack is usually prevented using an XOR-tree, while internal wire selective failure is prevented by various of ways, for example cut-and-choose and input recovery.

We argue that such selective failure attacks launched by $\mathsf{P_A}$ does not help $\mathsf{P_A}$ learn any information in our protocol. The key observation is that all wires, including input wires and internal wires are masked with some random masks (λ values) not known to $\mathsf{P_A}$. Therefore the best that $\mathsf{P_A}$ could learn is masked wire values, which appears random to $\mathsf{P_A}$.

Correctness of the garbled circuit. Note that the garbled circuit in our protocol is not guaranteed to be computed correctly (This does not lead to an attack as explained above). However $\mathsf{P_B}$ is still able to evaluate the circuit if $\mathsf{P_B}$ does not abort. The key reason is that all permutation bits in the truth table are masked. Therefore, $\mathsf{P_A}$ cannot change the logic of the garbled table without breaking an IT-MAC.

5.2 The Main Proof

Theorem 5.1. *The protocol in Figure 2 securely instantiate $\mathcal{F}_{\mathbf{2pc}}$ in the $\mathcal{F}_{\mathsf{Pre}}$-hybrid model with security* $\mathsf{negl}(\kappa)$

Proof. We consider separately the case where $\mathsf{P_A}$ or $\mathsf{P_B}$ is malicious.

Malicious $\mathsf{P_A}$. Let $\mathcal{A}$ be an adversary corrupting $\mathsf{P_A}$. We construct a simulator $\mathcal{S}$ that runs $\mathcal{A}$ as a subroutine and plays the role of $\mathsf{P_A}$ in the ideal world involving an ideal functionality $\mathcal{F}$ evaluating f. $\mathcal{S}$ is defined as follows.

1-4 $\mathcal{S}$ interacts with $\mathcal{A}$ acting as an honest $\mathsf{P_B}$.

5 $\mathcal{S}$ interacts with $\mathcal{A}$ acting as an honest $\mathsf{P_B}$ using input $y = 0$.

6 $\mathcal{S}$ interacts with $\mathcal{A}$ acting as an honest $\mathsf{P_B}$. For each wire $w \in \mathcal{I}_1$ $\mathcal{S}$ receives $x_w \oplus \lambda_w$ and computes $x_w = (x_w \oplus \lambda_w) \oplus r_w \oplus s_w$. $\mathcal{S}$ sends x to $\mathcal{F}_{\mathbf{2pc}}$.

7-8 $\mathcal{S}$ interacts with $\mathcal{A}$ acting as an honest $\mathsf{P_B}$. If an honest $\mathsf{P_B}$ would abort, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and aborts; otherwise $\mathcal{S}$ sends $\mathsf{continue}$ to $\mathcal{F}_{\mathbf{2pc}}$.

We now show that the joint distribution over the outputs of $\mathcal{A}$ and the honest $\mathsf{P_B}$ in the real world is indistinguishable from the joint distribution over the outputs of $\mathcal{S}$ and $\mathsf{P_B}$ in the ideal world.

Hybrid$_1$. Same as the hybrid-world protocol, where $\mathcal{S}$ plays the role of an honest $\mathsf{P_B}$ using the actual input y.

Hybrid$_2$. Same as **Hybrid$_1$**, except that in step 6, for each wire $w \in \mathcal{I}_1$ $\mathcal{S}$ receives $x_w \oplus \lambda_w$ and computes $x_w = (x_w \oplus \lambda_w) \oplus r_w \oplus s_w$. $\mathcal{S}$ sends x to $\mathcal{F}_{\mathbf{2pc}}$. If an honest $\mathsf{P_B}$ would abort, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and aborts; otherwise $\mathcal{S}$ sends $\mathsf{continue}$ to $\mathcal{F}_{\mathbf{2pc}}$.

The view of the two **Hybrid**s are exactly the same. According to Lemma 5.1, $\mathsf{P_B}$ will learn the same output in both **Hybrid**s with all but negligible probability.

Hybrid$_3$. Same as **Hybrid$_2$**, except that $\mathcal{S}$ computes $\{s_w\}_{w\in\mathcal{I}_2}$ as follows: $\mathcal{S}$ first randomly pick $\{u_w\}_{w\in\mathcal{I}_2}$, and then computes $s_w := u_w \oplus y_w$.

The two **Hybrid**s have exactly the same view.

Hybrid$_4$. Same as **Hybrid$_3$**, except that $\mathcal{S}$ uses $y = 0$ as inputs throughout the protocol.

Note that although the distribution of y in **Hybrid$_3$** and **Hybrid$_4$** are different, the distribution of $s_w \oplus y_w$ are exactly the same. The view of the two **Hybrid**s are therefore the same, we will show that $\mathsf{P_B}$ aborts with the same probability in two **Hybrid**s.

Observe that the only place where $\mathsf{P_B}$'s abort can possibly depends on y is in step 7(b). However, this abort depends on which row is selected to decrypt, that is the value of $\lambda_\alpha \oplus z_\alpha$ and $\lambda_\beta \oplus z_\beta$, which are chosen independently random in both **Hybrid**s.

As **Hybrid$_4$** is the ideal-world execution, this completes the proof for a malicious $\mathsf{P_A}$.

Malicious $\mathsf{P_B}$. Let $\mathcal{A}$ be an adversary corrupting $\mathsf{P_B}$. We construct a simulator $\mathcal{S}$ that runs $\mathcal{A}$ as a subroutine and plays the role of $\mathsf{P_B}$ in the ideal world involving an ideal functionality $\mathcal{F}$ evaluating f. $\mathcal{S}$ is defined as follows.

1-4 $\mathcal{S}$ interacts with $\mathcal{A}$ acting as an honest $\mathsf{P_A}$ and plays the functionality of $\mathcal{F}_{\mathsf{Pre}}$. If an honest $\mathsf{P_A}$ would abort, $\mathcal{S}$ output whatever $\mathcal{A}$ outputs and aborts.

5 $\mathcal{S}$ interacts with $\mathcal{A}$ acting as an honest $\mathsf{P_A}$, receives $y_w \oplus \lambda_w$ from $\mathcal{A}$, and computes $y_w := (y_w \oplus \lambda_w) \oplus s_w \oplus r_w$, where s_w, r_w are values $\mathcal{S}$ used when playing the role of $\mathcal{F}_{\mathsf{Pre}}$. $\mathcal{S}$ sends y to $\mathcal{F}_{\mathbf{2pc}}$, which sends $z = f(x, y)$ to $\mathcal{S}$.

6 $\mathcal{S}$ interacts with $\mathcal{A}$ acting as an honest $\mathsf{P_A}$ using input $x = 0$. If an honest $\mathsf{P_A}$ would abort, $\mathcal{S}$ output whatever $\mathcal{A}$ outputs and aborts.

8 $\mathcal{S}$ computes $z' = f(0, y)$. For each $w \in \mathcal{O}$, if $z'_w = z_w$, $\mathcal{S}$ sends $(r_w, \mathsf{M}[r_w])$; otherwise, $\mathcal{S}$ sends $(r_w \oplus 1, \mathsf{M}[r_w] \oplus \Delta_\mathsf{B})$.

We now show that the joint distribution over the outputs of $\mathcal{A}$ and the honest $\mathsf{P_A}$ in the real world is indistinguishable from the joint distribution over the outputs of $\mathcal{S}$ and $\mathsf{P_A}$ in the ideal world.

Hybrid$_1$. Same as the hybrid-world protocol, where $\mathcal{S}$ plays the role of an honest $\mathsf{P_A}$ using the actual input x.

Hybrid$_2$. Same as **Hybrid$_1$**, except that, in step 5, $\mathcal{S}$ receives $y_w \oplus \lambda_w$ from $\mathcal{A}$, and computes $y_w := (y_w \oplus \lambda_w) \oplus s_w \oplus r_w$, where s_w, r_w are values $\mathcal{S}$ used when playing the role of $\mathcal{F}_{\mathsf{Pre}}$. $\mathcal{S}$ then sends y to $\mathcal{F}_{\mathbf{2pc}}$, and receives $z = f(x, y)$.

$\mathsf{P_A}$ does not have output; further the view of $\mathcal{A}$ does not change between two **Hybrid**s.

Hybrid$_3$. Same as **Hybrid$_2$**, except that in step 6, $\mathcal{S}$ uses $x = 0$ as input and in step 8, $\mathcal{S}$ computes $z' = f(0, y)$. For each $w \in \mathcal{O}$, if $z'_w = z_w$, $\mathcal{S}$ sends $(r_w, \mathsf{M}[r_w])$; otherwise, $\mathcal{S}$ sends $(r_w \oplus 1, \mathsf{M}[r_w] \oplus \Delta_{\mathsf{B}})$.

$\mathcal{A}$ has no knowledge of r_w, therefore r_w and $r_w \oplus 1$ are indistinguishable.

Note that since $\mathcal{S}$ uses different values for x between two **Hybrid**s, we also need to show that the garbled rows $\mathsf{P_B}$ opened are indistinguishable between two **Hybrid**s. According to Lemma 5.2, $\mathsf{P_B}$ is able to open only one garble rows in each garbled table $G_{\gamma,i}$. Therefore, given that $\{\lambda_w\}_{w \in \mathcal{I}_1 \cup \mathcal{W}}$ values are not known to $\mathsf{P_B}$, masked values and garbled keys are indistinguishable between two **Hybrid**s.

As **Hybrid$_3$** is the ideal-world execution, the proof is complete. □

Lemma 5.1. *Consider an $\mathcal{A}$ corrupting $\mathsf{P_A}$ and denote $x_w := (x_w \oplus \lambda_w) \oplus s_w \oplus r_w$, where $x_w \oplus \lambda_w$ is the value $\mathcal{A}$ sent to $\mathsf{P_B}$, s_w, r_w are the values from $\mathcal{F}_{\mathsf{Pre}}$. With probability all but negligible, $\mathsf{P_B}$ either aborts or learns $z = f(x, y)$.*

Proof. Define z^*_w as the correct wire values computed using x defined above and y, z_w as the actually wire values $\mathsf{P_B}$ holds in the evaluation.

We will first show that $\mathsf{P_B}$ learns $\{z^w \oplus \lambda_w = z^*_w \oplus \lambda_w\}_{w \in \mathcal{O}}$ by induction on topology of the circuit.

Base step: It is obvious that $\{z^*_w \oplus \lambda_w = z_w \oplus \lambda_w\}_{w \in \mathcal{I}_1 \cup \mathcal{I}_2}$, unless $\mathcal{A}$ is able to forge an IT-MAC.

Induction step: Now we show that for a gate $(\alpha, \beta, \gamma, T)$, if $\mathsf{P_B}$ has $\{z^*_w \oplus \lambda_w = z_w \oplus \lambda_w\}_{w \in \{\alpha, \beta\}}$, then $\mathsf{P_B}$ also obtains $z^*_\gamma \oplus \lambda_\gamma = z_\gamma \oplus \lambda_\gamma$.

- $T = \oplus$: It is true according to the following: $z^*_\gamma \oplus \lambda_\gamma = (z^*_\alpha \oplus \lambda_\alpha) \oplus (z^*_\beta \oplus \lambda_\beta) = (z_\alpha \oplus \lambda_\alpha) \oplus (z_\beta \oplus \lambda_\beta) z_\gamma \oplus \lambda_\gamma$
- $T = \wedge$: According to the protocol, $\mathsf{P_B}$ will open the garbled row defined by $i := 2(z_\alpha \oplus \lambda_\alpha) + (z_\beta \oplus \lambda_\beta)$. If $\mathsf{P_B}$ learns $z_\gamma \oplus \lambda_\gamma \neq z^*_\gamma \oplus \lambda_\gamma$, then it means that $\mathsf{P_B}$ learns $r^*_{\gamma,i} \neq r_{\gamma,i}$. However, this would mean that $\mathcal{A}$ forge a valid IT-MAC, happening with negligible probability.

Now we know that $\mathsf{P_B}$ learns correct masked output. $\mathsf{P_B}$ can therefore learn correct output $f(x, y)$ unless $\mathcal{A}$ is able to flip $\{r_w\}_{w \in \mathcal{O}}$, which, again, happens with negligible probability. □

Lemma 5.2. *Consider an $\mathcal{A}$ corrupting $\mathsf{P_B}$, with negligible, probability, $\mathsf{P_B}$ learns both garbled keys for some wire.*

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

Honest parties: The box picks random $[x_1]_\mathsf{A}, [y_1]_\mathsf{A}, [z_1]_\mathsf{A}$, and $[x_2]_\mathsf{B}, [y_2]_\mathsf{B}, [z_2]_\mathsf{B}$, such that $(x_1 \oplus x_2) \wedge (y_1 \oplus y_2) = z_1 \oplus z_2$.

Corrupted parties:

1. A corrupted $\mathsf{P_A}$ gets to choose all its randomness. Further, it can send g to the box trying to guess x_2. If $g \neq x_2$ the box output fail and terminate, otherwise the box process as normal.
2. A corrupted $\mathsf{P_B}$ gets to choose all its randomness. Further, it can send g to the box trying to guess x_1. If $g \neq x_1$ the box output fail and terminate, otherwise the box process as normal.

Global Key Queries: The adversary at any point can send some (p, Δ') and will be told if $\Delta' = \Delta_p$.

Figure 3: Functionality $\mathcal{F}_{\mathsf{LaAND}}$ for leaky AND triple generation.

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

Honest parties: The box picks random $[x_1]_\mathsf{A}, [y_1]_\mathsf{A}, [z_1]_\mathsf{A}$, and $[x_2]_\mathsf{B}, [y_2]_\mathsf{B}, [z_2]_\mathsf{B}$, such that $(x_1 \oplus x_2) \wedge (y_1 \oplus y_2) = z_1 \oplus z_2$.

Corrupted parties: A corrupted $\mathsf{P_A}$ gets to choose all its randomness.

Global Key Queries: The adversary at any point can send some (p, Δ') and will be told if $\Delta' = \Delta_p$.

Figure 4: Functionality $\mathcal{F}_{\mathsf{aAND}}$ for generating AND triples

Proof. The proof is very similar to the proof of security for garbled circuits in the semi-honest setting.

Base step: $\mathsf{P_B}$ can only learn one garbled keys for each input wire, since $\mathsf{P_A}$ only sends one garbled wire, and $\mathsf{P_B}$ cannot learn Δ_A in the protocol.

Induction step: It is obvious that $\mathsf{P_B}$ cannot learn the other label for an XOR gate and we will focus on AND gates.

Note that $\mathsf{P_B}$ only learns one garbled keys for input wire α and β. However, each row is encrypted using different combinations of $\{L_{\alpha,b}\}_{b\in\{0,1\}}$ and $\{\mathsf{L}_{\beta,b}\}_{b\in\{0,1\}}$. In order for $\mathsf{P_B}$ to open two rows in the garbled table, $\mathsf{P_B}$ needs to learn both garbled keys for some input wire, which contradict with assumptions in the induction step. $\square$

6 Improved TinyOT protocol

In this section, we describe an improvement to the TinyOT protocol. For a bucket size of $B = \frac{\rho}{\log|\mathcal{C}|} + 1$, the original protocol requires $14B + 2$ authenticated bits for each AND gate. In the following, we will introduce an improved version where only $6B$ authenticated bits are needed for each AND gate. For a circuit of size 2^{20}, with $\rho = 40$, this is an improvement of $2.4\times$.

Assuming that two parties hold $[x_1]_\mathsf{A}, [y_1]_\mathsf{A}, [x_2]_\mathsf{B}, [y_2]_\mathsf{B}$. In the original TinyOT protocol, to compute $(x_1 \oplus x_2)(y_1 \oplus y_2)$, $\mathsf{P_A}$ and $\mathsf{P_B}$ compute $[x_1y_1]_\mathsf{A}$, $[x_2y_2]_\mathsf{B}$, $[x_1y_2 + r]_\mathsf{A}$ and $[x_2y_1 + r]_\mathsf{B}$ separately , with some random $r \in \{0, 1\}$, using various authenticated constructions proposed in their paper. Computing each entry separately incurs a lot of unnecessary cost. We observe that it is possible to compute a whole AND gate directly. Similar to the original TinyOT protocol, we

Protocol Π_{LaAND}

1. $\mathsf{P_A}$ and $\mathsf{P_B}$ obtain random authenticated bits $[x_1]_\mathsf{A}, [y_1]_\mathsf{A}, [z_1]_\mathsf{A}, [x_2]_\mathsf{B}, [y_2]_\mathsf{B}, [r]_\mathsf{B}$.
2. $\mathsf{P_A}$ parses $\mathsf{K}[x_2] := [x_2]_\mathsf{B}, \mathsf{K}[y_2] := [y_2]_\mathsf{B}$ and sends the following four bits to $\mathsf{P_B}$.
$$G_{0,0} := \mathsf{Lsb}(H(\mathsf{K}[x_2], \mathsf{K}[y_2])) \oplus (0 \oplus x_1) \wedge (0 \oplus y_1) \oplus z_1$$
$$G_{1,0} := \mathsf{Lsb}(H(\mathsf{K}[x_2] \oplus \Delta_\mathsf{A}, \mathsf{K}[y_2])) \oplus (1 \oplus x_1) \wedge (0 \oplus y_1) \oplus z_1$$
$$G_{0,1} := \mathsf{Lsb}(H(\mathsf{K}[x_2], \mathsf{K}[y_2] \oplus \Delta_\mathsf{A})) \oplus (0 \oplus x_1) \wedge (1 \oplus y_1) \oplus z_1$$
$$G_{1,1} := \mathsf{Lsb}(H(\mathsf{K}[x_2] \oplus \Delta_\mathsf{A}, \mathsf{K}[y_2] \oplus \Delta_\mathsf{A})) \oplus (1 \oplus x_1) \wedge (1 \oplus y_1) \oplus z_1$$
3. $\mathsf{P_B}$ parses $(x_2, \mathsf{M}[x_2]) := [x_2]_\mathsf{B}, (y_2, \mathsf{M}[y_2]) := [y_2]_\mathsf{B}$ and computes $z_2 := \mathsf{Lsb}(H(\mathsf{M}[x_2], \mathsf{M}[y_2])) \oplus G_{x_2,y_2}$. $\mathsf{P_B}$ announces $d := r \oplus z_2$ to $\mathsf{P_A}$. Two parties compute $[z_2]_\mathsf{B} = [r]_\mathsf{B} \oplus d$.
4. $\mathsf{P_B}$ checks the correctness as follows:
   (a) $\mathsf{P_B}$ computes:
   $$T_0 := H(\mathsf{K}[x_1], \mathsf{K}[z_1] \oplus z_2\Delta_\mathsf{B})$$
   $$U_0 := T_0 \oplus H(\mathsf{K}[x_1] \oplus \Delta_\mathsf{B}, \mathsf{K}[y_1] \oplus \mathsf{K}[z_1] \oplus (y_2 \oplus z_2)\Delta_\mathsf{B})$$
   $$T_1 := H(\mathsf{K}[x_1], \mathsf{K}[y_1] \oplus \mathsf{K}[z_1] \oplus (y_2 \oplus z_2)\Delta_\mathsf{B})$$
   $$U_1 := T_1 \oplus H(\mathsf{K}[x_1] \oplus \Delta_\mathsf{B}, \mathsf{K}[z_1] \oplus z_2\Delta_\mathsf{B})$$
   (b) $\mathsf{P_B}$ sends U_{x_2} to $\mathsf{P_A}$.
   (c) $\mathsf{P_A}$ randomly picks a κ-bit string R and computes
   $$V_0 := H(\mathsf{M}[x_1], \mathsf{M}[z_1]) \qquad V_1 := H(\mathsf{M}[x_1], \mathsf{M}[z_1] \oplus \mathsf{M}[y_1])$$
   $$W_{0,0} := H(\mathsf{K}[x_2]) \oplus V_0 \oplus R \qquad W_{0,1} := H(\mathsf{K}[x_2] \oplus \Delta_\mathsf{A}) \oplus V_1 \oplus R$$
   $$W_{1,0} := H(\mathsf{K}[x_2]) \oplus V_1 \oplus U \oplus R \qquad W_{1,1} := H(\mathsf{K}[x_2] \oplus \Delta_\mathsf{A}) \oplus V_0 \oplus U \oplus R$$
   (d) $\mathsf{P_A}$ sends $W_{x_1,0}, W_{x_1,1}$ to $\mathsf{P_B}$ and sends R to $\mathcal{F}_{\mathsf{EQ}}$.
   (e) $\mathsf{P_B}$ computes $R' := W_{x_1,x_2} \oplus H(\mathsf{M}[x_2]) \oplus T_{x_2}$ and sends R' to $\mathcal{F}_{\mathsf{EQ}}$.
5. $\mathsf{P_A}$ checks the correctness as follows:
   (a) $\mathsf{P_A}$ computes:
   $$T_0 := H(\mathsf{K}[x_2], \mathsf{K}[z_2] \oplus z_1\Delta_\mathsf{A})$$
   $$U_0 := T_0 \oplus H(\mathsf{K}[x_2] \oplus \Delta_\mathsf{A}, \mathsf{K}[y_2] \oplus \mathsf{K}[z_2] \oplus (y_1 \oplus z_1)\Delta_\mathsf{A})$$
   $$T_1 := H(\mathsf{K}[x_2], \mathsf{K}[y_2] \oplus \mathsf{K}[z_2] \oplus (y_1 \oplus z_1)\Delta_\mathsf{A})$$
   $$U_1 := T_1 \oplus H(\mathsf{K}[x_2] \oplus \Delta_\mathsf{A}, \mathsf{K}[z_2] \oplus z_1\Delta_\mathsf{A})$$
   (b) $\mathsf{P_A}$ sends U_{x_1} to $\mathsf{P_B}$.
   (c) $\mathsf{P_B}$ randomly picks a κ-bit string R and computes
   $$V_0 := H(\mathsf{M}[x_2], \mathsf{M}[z_2]) \qquad V_1 := H(\mathsf{M}[x_2], \mathsf{M}[z_2] \oplus \mathsf{M}[y_2])$$
   $$W_{0,0} := H(\mathsf{K}[x_1]) \oplus V_0 \oplus R \qquad W_{0,1} := H(\mathsf{K}[x_1] \oplus \Delta_\mathsf{B}) \oplus V_1 \oplus R$$
   $$W_{1,0} := H(\mathsf{K}[x_1]) \oplus V_1 \oplus U \oplus R \qquad W_{1,1} := H(\mathsf{K}[x_1] \oplus \Delta_\mathsf{B}) \oplus V_0 \oplus U \oplus R$$
   (d) $\mathsf{P_B}$ sends $W_{x_2,0}, W_{x_2,1}$ to $\mathsf{P_A}$ and sends R to $\mathcal{F}_{\mathsf{EQ}}$,
   (e) $\mathsf{P_A}$ computes $R' := W_{x_2,x_1} \oplus H(\mathsf{M}[x_1]) \oplus T_{x_1}$ and sends R' to $\mathcal{F}_{\mathsf{EQ}}$.

Figure 5:

propose a "leaky AND" protocol (Π_{LaAND}), where adversary is allowed to perform selective failure attack on one input, and later use bucketing and combining to eliminate such leakage (Π_{aAND}). In the following, we will first discuss intuition of the protocol. The full protocol description is in Figure 5 and Figure 6.

Compute the triple in the honest case. The first step of the protocol is to generate the triple securely assuming that both parties are honest. Since x_1, y_1, z_1, x_2, y_2 are all random, we just need

Protocol Π_{aAND}

1. $\mathsf{P_A}$ and $\mathsf{P_B}$ call $\mathcal{F}_{\mathsf{LaAND}}$ $\ell' = \ell B$ times and obtains $\{[x_1^i]_{\mathsf{A}}, [y_1^i]_{\mathsf{A}}, [z_1^i]_{\mathsf{A}}, [x_2^i]_{\mathsf{B}}, [y_2^i]_{\mathsf{B}}, [z_2^i]_{\mathsf{B}}\}_{i=1}^{\ell'}$.
2. $\mathsf{P_A}$ and $\mathsf{P_B}$ randomly partition all objects into ℓ buckets, each with B objects.
3. For each bucket, two parties combine B Leaky ANDs into one non-leaky AND. To combine two leaky ANDs, namely $([x_1']_{\mathsf{A}}, [y_1']_{\mathsf{A}}, [z_1']_{\mathsf{A}}, [x_2']_{\mathsf{B}}, [y_2']_{\mathsf{B}}, [z_2']_{\mathsf{B}})$ and $[x_1'']_{\mathsf{A}}, [y_1'']_{\mathsf{A}}, [z_1'']_{\mathsf{A}}, [x_2'']_{\mathsf{B}}, [y_2'']_{\mathsf{B}}, [z_2'']_{\mathsf{B}}$
   (a) Two parties reveal $d' := y_1' \oplus y_1'', d'' = y_2' \oplus y_2''$ with their MAC checked, and compute $d := d' \oplus d''$.
   (b) Set $[x_1]_{\mathsf{A}} := [x_1']_{\mathsf{A}} \oplus [x_1'']_{\mathsf{A}}$, $[x_2]_{\mathsf{B}} := [x_2']_{\mathsf{B}} \oplus [x_2'']_{\mathsf{B}}$, $[y_1]_{\mathsf{A}} := [y_1']_{\mathsf{A}}$, $[y_2]_{\mathsf{A}} := [y_2']_{\mathsf{A}}$, $[z_1]_{\mathsf{A}} := [z_1']_{\mathsf{A}} \oplus [z_1'']_{\mathsf{A}} \oplus d[x_1'']_{\mathsf{A}}$, $[z_2]_{\mathsf{B}} := [z_2']_{\mathsf{B}} \oplus [z_2'']_{\mathsf{B}} \oplus d[x_2'']_{\mathsf{B}}$.

   Two parties iterate all B leaky objects, by taking the resulted object and combine with the next element.

Figure 6: Protocol Π_{aAND} instantiating $\mathcal{F}_{\mathsf{aAND}}$.

$\mathsf{P_B}$ to learn $z_2 = (x_1 \oplus x_2) \wedge (y_1 \oplus y_2) \oplus z_1$. Our idea is to generate a garbled table for AND. We observe that if we treat $\mathsf{K}[x_2], \mathsf{K}[y_2]$ as zero garbled label, then $\mathsf{M}[x_2], \mathsf{M}[y_2]$ are garbled labels representing the underlying values, that is we do not need oblivious transfer to let $\mathsf{P_B}$ obtain the label. Further, authenticity is not needed in our case, which means we do not need $\mathsf{P_B}$ to learn the whole label, as long as $\mathsf{P_B}$ learns the output. Inspired by these, our construction only requires 4 bits in order for $\mathsf{P_B}$ to learn z_2 (step 1 to 3 in Figure 5).

Verifying the correctness. The above steps are not enough for malicious security: a malicious $\mathsf{P_A}$ can cheat by sending incorrect garbled tables and a malicious $\mathsf{P_B}$ can annouce a incorrect d in step 3. Therefore, both parties needs to check the correctness of the output. In the protocol, we designed a verification protocol that check the correctness while allowing a malicious party to perform a selective failure attack on x values.

The initial idea was to adopt the check from TinyOT to our case. If $x_2 \oplus x_1 = 0$, then we want to check that $z_2 = z_1$; if $x_2 \oplus x_1 = 1$, then to check $y_1 \oplus z_1 = y_2 \oplus z_2$. However, an obvious problem is that no party knows the value of $x_1 \oplus x_2$. To solve this problem, when $\mathsf{P_B}$ checks the correctness, we let $\mathsf{P_B}$ construct the checking depending on the value of x_2. $\mathsf{P_A}$ will perform the checking twice, as if x_2 is 0 and 1.

For example, using notations in the protocol, when $x_1 = 0$, $\mathsf{P_A}$ computes V_0, V_1. $\mathsf{P_A}$ and $\mathsf{P_B}$ should have performed an equality check between V_{x_2} and T_{x_2}. All different cases (depending on the value of x_1 and x_2) are summarized in the following table.

	$x_1 = 0$	$x_1 = 1$
$x_2 = 0$	$V_0 = T_0$	$V_0 \oplus U_0 = T_0$
$x_2 = 1$	$V_1 = T_1$	$V_1 \oplus U_1 = T_1$

However, $\mathsf{P_A}$ should not learn x_2, while $\mathsf{P_B}$ should not learn $V_{1 \oplus x_2}$. One idea is to let $\mathsf{P_A}$ "encrypt" the response (V_0, V_1) such that $\mathsf{P_B}$ can only learn the response for the value of x_2 (V_{x_2}), then $\mathsf{P_B}$ can compare locally. (This is possible because $\mathsf{P_B}$'s bit x_2 is authenticated by $\mathsf{P_A}$). However, the problem is that $\mathsf{P_A}$ is not able to learn the outcome of the comparison. To solve this, we let $\mathsf{P_A}$ send encrypted $V_0 \oplus R$ and $V_1 \oplus R$ for some random R such that $\mathsf{P_B}$ learns $V_{x_2} \oplus R$, and learn R from it. Now $\mathsf{P_A}$ and $\mathsf{P_B}$ can check the equality on R using the $\mathcal{F}_{\mathsf{EQ}}$ functionality in the TinyOT paper that allows both parties get the outcome. Note that this allows $\mathsf{P_A}$ to perform an additional selective failure attack on x_2, by sending some corrupted encrypted values. This does not introduce additional leakage, since x_2 is allowed to be learnt by $\mathcal{A}$ anyway. Now $\mathcal{A}$ is allowed to guess x_2 twice, once in step 4 and once in step 5. If the guesses are inconsistent, it is guaranteed to abort.

Combining leaky ANDs. The above check is vulnerable to selective failure attack, from which a malicious party can learn the value of x_1/x_2 with a risk of caught with one-half probability. In order to get rid of the leakage, bucketing is performed similar to TinyOT. Here, the key is to devise a way to combine leaky objects. Assuming that two triple are $([x_1']_{\mathsf{A}}, [y_1']_{\mathsf{A}}, [z_1']_{\mathsf{A}}, [x_2']_{\mathsf{B}}, [y_2']_{\mathsf{B}}, [z_2']_{\mathsf{B}})$ and $[x_1'']_{\mathsf{A}}, [y_1'']_{\mathsf{A}}, [z_1'']_{\mathsf{A}}, [x_2'']_{\mathsf{B}}, [y_2'']_{\mathsf{B}}, [z_2'']_{\mathsf{B}}$. Note that for each triple, only x_1, x_2 can be leaked. Therefore, one natural way is to set $[x_1]_{\mathsf{A}} := [x_1']_{\mathsf{A}} \oplus [x_1'']_{\mathsf{A}}, [x_2]_{\mathsf{B}} := [x_2']_{\mathsf{B}} \oplus [x_2'']_{\mathsf{B}}$. By doing this, $[x_1]_{\mathsf{A}}, [x_2]_{\mathsf{B}}$ are non-leaky as long as one triple is non-leaky. We can also set $[y_1]_{\mathsf{A}} := [y_1']_{\mathsf{A}}, [y_2]_{\mathsf{B}} := [y_2']_{\mathsf{B}}$ and reveal the bit $d := y_1' \oplus y_2' \oplus y_1'' \oplus y_2''$, since y's bits are all private. Now observe that

$$\begin{aligned}
(x_1 \oplus x_2)(y_1 \oplus y_2) &= (x_1' \oplus x_2' \oplus x_1'' \oplus x_2'')(y_1' \oplus y_2') \\
&= (x_1' \oplus x_2')(y_1' \oplus y_2') \oplus (x_1'' \oplus x_2'')(y_1' \oplus y_2') \\
&= (x_1' \oplus x_2')(y_1' \oplus y_2') \oplus (x_1'' \oplus x_2'')(y_1'' \oplus y_2'') \oplus (x_1'' \oplus x_2'')(y_1' \oplus y_2' \oplus y_1'' \oplus y_2'') \\
&= (z_1' \oplus z_2') \oplus (z_1'' \oplus z_2'') \oplus d(x_1'' \oplus x_2'') \\
&= (z_1' \oplus z_1'' \oplus dx_1'') \oplus (z_2' \oplus z_2'' \oplus dx_2'')
\end{aligned}$$

Therefore, we could just set $[z_1]_{\mathsf{A}} := [z_1']_{\mathsf{A}} \oplus [z_1'']_{\mathsf{A}} \oplus d[x_1'']_{\mathsf{A}}$, $[z_2]_{\mathsf{A}} := [z_2']_{\mathsf{A}} \oplus [z_2'']_{\mathsf{A}} \oplus d[x_2'']_{\mathsf{A}}$. The security of this bucketing and merging can be proved as in [NNOB12, Appendix I].

6.1 Proof Sketch

In the following, we will discuss from a high-level view how the proof works for the new TinyOT protocol. We will focus on the security of Π_{LaAND} protocol, since the security of Π_{aAND} is fairly straightforward given the proof in the original paper [NNOB12].

Lemma 6.1. *The protocol in Figure 5 securely implements the functionality in Figure 3 against corrupted* $\mathsf{P_A}$ *in the* $(\mathcal{F}_{\mathsf{abit}}, \mathcal{F}_{\mathsf{EQ}})$*-Hybrid model.*

Proof. We will construct a simulator as follows:

1 $\mathcal{S}$ interacts with $\mathcal{A}$ and receive $(x_1, \mathsf{M}[x_1]), (y_1, \mathsf{M}[y_1]), (z_1, \mathsf{M}[z_1]), \mathsf{K}[x_2], \mathsf{K}[y_2], \mathsf{K}[r], \Delta_{\mathsf{A}}$ that $\mathcal{A}$ sent to $\mathcal{F}_{\mathsf{abit}}$. $\mathcal{S}$ picks a random bit s, sets $\mathsf{K}[z_2] := \mathsf{K}[r] \oplus s\Delta_{\mathsf{A}}$, and sends $(x_1, \mathsf{M}[x_1]), (y_1, \mathsf{M}[y_1]), (z_1, \mathsf{M}[z_1]), \mathsf{K}[x_2], \mathsf{K}[y_2], \mathsf{K}[z_2], \Delta_{\mathsf{A}})$ to $\mathcal{F}_{\mathsf{LaAND}}$, which sends $(x_2, \mathsf{M}[x_2]), (y_2, \mathsf{M}[y_2])(z_2, \mathsf{M}[z_2]), \mathsf{K}[x_1], \mathsf{K}[y_1], \mathsf{K}[z_1], \Delta_{\mathsf{B}})$ to $\mathsf{P_B}$.

2-3 $\mathcal{S}$ randomly picks one row of $G_{i,j}$ and check its correctness. If it is not computed correctly, $\mathcal{S}$ aborts; otherwise, $\mathcal{S}$ annouce $s \oplus z_2$.

4 $\mathcal{S}$ sends a random U^* to $\mathcal{A}$, and receives some W_0, W_1 and computes some R_0, R_1, such that, if $x_1 = 0$, $W_0 := H(\mathsf{K}[x_2]) \oplus V_0 \oplus R_1, W_1 := H(\mathsf{K}[x_2] \oplus \Delta_{\mathsf{A}}) \oplus V_1 \oplus R_2$; otherwise, $W_0 := H(\mathsf{K}[x_2]) \oplus V_1 \oplus U^* \oplus R_1$ and $W_1 := H(\mathsf{K}[x_2] \oplus \Delta_{\mathsf{A}}) \oplus V_0 \oplus U^* \oplus R_2$.

   $\mathcal{S}$ also obtains R that $\mathcal{A}$ sent to $\mathcal{F}_{\mathsf{EQ}}$. If R does not equal to either R_0 or R_1, $\mathcal{S}$ aborts; otherwise $\mathcal{S}$ computes g_1 such that $R \neq R_{g_1}$ for some $g_1 \in \{0, 1\}$.

5 $\mathcal{S}$ receives U, picks random W_0^*, W_1^* and sends them to $\mathcal{A}$. $\mathcal{S}$ obtains R' that $\mathcal{A}$ sent to $\mathcal{F}_{\mathsf{EQ}}$.

   - If both U, R' are honestly computed, $\mathcal{S}$ proceeds as normal.

- If U is not honestly computed and that $R' = W^*_{x_1} \oplus H(\mathsf{M}[x_1]) \oplus T_{x_1}$ is honestly computed, $\mathcal{S}$ set $g_2 = 0$
- If either of the following is true: 1) $x_1 = 0$ and $R' = W^*_{x_1} \oplus H(\mathsf{M}[x_1]) \oplus U \oplus H(\mathsf{K}[x_1] \oplus \Delta_{\mathsf{B}}, \mathsf{K}[y_1] \oplus (y_2 \oplus z_2)\Delta_{\mathsf{B}})$; 2) $x_1 = 1$ and $R' = W^*_{x_1} \oplus H(\mathsf{M}[x_1]) \oplus U \oplus H(\mathsf{K}[x_1] \oplus \Delta_{\mathsf{B}}, \mathsf{K}[z_1] \oplus z_2\Delta_{\mathsf{B}})$, $\mathcal{S}$ sets $g_2 = 1$.
- Otherwise $\mathcal{S}$ aborts.

6 If $g_1 \neq g_2$, $\mathcal{S}$ aborts; otherwise, $\mathcal{S}$ sends g_1 to $\mathcal{F}_{\mathsf{LaAND}}$. If $\mathcal{F}_{\mathsf{LaAND}}$ abort, $\mathcal{S}$ aborts.

Note that the first 3 steps are perfect simulation. In step 4, U^* is sent in the simulation, while U_{x_2} is sent. This is a perfect simulation unless both of the input to Random Oracle in U_{x_2} get queried. This does not happen during the protocol, since Δ_{B} in not known to $\mathcal{A}$. In step 5, W^*_0, W^*_1 are sent in the simulation, while $W_{x_2,0}, W_{x_2,0}$ are sent in the real protocol. This is also a perfect simulation unless $\mathsf{P_A}$ gets Δ_{B}: both R and one of $H(\mathsf{K}[x_1])$ and $H(\mathsf{K}[x_1] \oplus \Delta_{\mathsf{B}})$ are random.

Another difference is that $\mathsf{P_B}$ always aborts in the simulation if G_{x_2,y_2} is not honestly computed. This is also the case in the real protocol unless $\mathcal{A}$ learns Δ_{B}.

□

Lemma 6.2. *The protocol in Figure 5 securely implements the functionality in Figure 3 against corrupted* $\mathsf{P_B}$ *in the* $(\mathcal{F}_{\mathsf{abit}}, \mathcal{F}_{\mathsf{EQ}})$*-Hybrid model.*

Proof. We will construct a simulator as follows:

1. $\mathcal{S}$ interacts with $\mathcal{A}$ and receive $(x_2, \mathsf{M}[x_2]), (y_2, \mathsf{M}[y_2]), (r, \mathsf{M}[r]), \mathsf{K}[x_1], \mathsf{K}[y_1], \mathsf{K}[z_1], \Delta_{\mathsf{B}}$ that $\mathcal{A}$ sent to $\mathcal{F}_{\mathsf{abit}}$. $\mathcal{S}$ picks a random bit s, sets $(z_2, \mathsf{M}[z_2]) := (r \oplus s, \mathsf{M}[z_2] \oplus s\Delta_{\mathsf{B}})$, and sends $(x_2, \mathsf{M}[x_2]), (y_2, \mathsf{M}[y_2]), (z_2, \mathsf{M}[z_2]), \mathsf{K}[x_1], \mathsf{K}[y_1], \mathsf{K}[z_1])$ to $\mathcal{F}_{\mathsf{LaAND}}$, which sends $(x_1, \mathsf{M}[x_1]), (y_1, \mathsf{M}[y_1]), (z_1, \mathsf{M}[z_1]), \mathsf{K}[x_2], \mathsf{K}[y_2], \mathsf{K}[z_2])$ to $\mathsf{P_B}$.

2-3 $\mathcal{S}$ sends $\mathcal{A}$ four random bits.

4-5 The simulation are the symmetric to the simulation for malicious $\mathsf{P_A}$.

The first three steps are perfect simulation; the proof for step 4 and 5 are the same as the proof for malicious $\mathsf{P_A}$ (with order of steps switched). □

7 Extensions and Optimizations

Reducing the size of the garbled table. In the original protocol, all MAC keys are κ-bit values, which may not be necessary. For ρ-bit statistical security, $\mathsf{M}[r_{00}]$ encrypted in step 4(d) only needs to be of size ρ bit. This reduces the size of a garbled table from 8κ bits to $4(\kappa + \rho)$ bits.

Partial garbled row reduction (PGGR). After applying the above optimization, the size of a garbled table is $4(\kappa + \rho + 1)$. However, we observe that randomness in $\mathsf{L}_{\gamma,0}$ are not utilized, which means we can potentially perform Garbled Row Reduction but only on part of the first row. In particular, instead of picking $\mathsf{L}_{\gamma,0}$ randomly, it will be set as $\mathsf{L}_{\gamma,0} = H(\mathsf{L}_{\alpha,0}, \mathsf{L}_{\beta,0}, \gamma, 0)[0 : \kappa]$, where $[0 : \kappa]$ means obtaining the lower κ bits.

Note that this optimization does not increase the round trip of the protocol, because there is no interaction needed at the time to send the garbled tables.

Bucket size	3	4	5
$\rho = 40$	280K	3.1K	320
$\rho = 64$	1.2G	780K	21K
$\rho = 80$	300G	32M	330K

Table 2: Least number of AND gates needed in the bucketing, for different bucket size and statistical security parameter.

Pushing computation to earlier phases. In our protocol, it requires online communication complexity $|\mathcal{I}|(\kappa + \rho) + |\mathcal{O}|\rho$. It is easy to eliminate the term $|\mathcal{O}|\rho$, by sending encryption of $\{(r_w, \mathsf{M}[r_w])\}_{w\in\mathcal{O}}$ and commitment of the key used in the encryption. There is no increase in communication, round trip or total computation. Further, IT-MACs in step 5 and step 6 can also be sent in the function independent phase. Further, IT-MAC associated with input masks can also be sent in the function dependent phase. The resulting online phase has communication $|\mathcal{I}|\kappa$.

Extending to a two-output protocol. Our protocol can be extended to a two-output version such that generator also gets an output. Denoting $\mathcal{O}_1$ as the set of output wire-indices for P_A, then after step 7, P_B learns $\{\mathsf{L}_{w,z\oplus\lambda_w}\}_{w\in\mathcal{O}_1}$. Instead of following step 8, P_B sends $\{s_w, \mathsf{M}[s_w], \mathsf{L}_{w,z\oplus\lambda_w}\}_{w\in\mathcal{O}_1}$ to P_A, who check the validity of s_w, computes λ_w, and computes z using $\mathsf{L}_{w,z\oplus\lambda_w}$ and λ_w. P_B is not able to obtain P_A's input, since the values are masked by some value unknown to P_B; P_B also cannot flip the output, which requires either knowing Δ_1 or forging an IT-MAC.

More TinyOT optimizations. In the following we will briefly discuss more optimizations when $\mathcal{F}_{\mathsf{Pre}}$ is instantiated using our TinyOT protocol.

1. For clarity, R was chosen randomly in Π_{LaAND}. It is possible to perform garbled row reduction, such that $W_{0,0}, W_{1,0}$ are zero. This saves two ciphertext per leaky AND.

2. For the value of R's and U's, only ρ bits of the values are needed to be sent.

8 Evaluation

We are planning to implement our protocol. In the following, we will discuss the communication complexity of our scheme compared to other schemes as well as the cost at each stage. Throughout this section, all numbers will be given with $\kappa = 128, \rho = 40$.

We count the communication of TinyOT based on optimization mentioned in previous sections. After applying the optimization from Nielsen et al. [NST17] for the authenticated bit, the communication for each authenticated bit is 21 bytes and the communication for authenticated AND is about $94B$ bytes from each party, where B is the bucket size.

In Table 2 we calculated the smallest number of gates needed in the TinyOT protocol in order to make the bucketing works. The calculation is based on the formula in Appendix B of their paper, which is tighter. In Table 3 we compare the communication complexity of our protocol with other related works. Similar to previous papers, only one way communication is counted. As we can see, our total communication is 3× to 6× less than Nielsen et al.'s protocol. Further, our cost with single execution is also twice less than the cost of Nielsen et al.'s with 1024 circuits. Note that

Protocol	#execution	Ind. Process	Dep. Process	Online
[RR16]	32	-	3.75 MB	25.76 kB
	128	-	2.5 MB	21.31 kB
	1024	-	1.56 MB	16.95 kB
[NST17]	1	14.94 MB	226.86 kB	16.13 kB
	32	8.74 MB	226.86 kB	16.13 kB
	128	7.22 MB	226.86 kB	16.13 kB
	1024	6.42 MB	226.86 kB	16.13 kB
This paper	1	2.56 MB	464.3 kB	4.1 kB
	32	2.56 MB	464.3 kB	4.1 kB
	128	1.92 MB	464.3 kB	4.1 kB
	1024	1.92 MB	464.3 kB	4.1 kB

Table 3: Comparison of communication with previous protocols for an AES circuit. Data are counted as the amount sent to evaluator. Total communication will be roughly doubled for [RR16] and this paper.

for protocols based on cut-and-choose, the total communication to send 40 AES garbled circuit is 8.7 MB, which is already higher than the total communication of ours in the single execution setting.

We also observe that our function dependent processing is higher than Nielsen et al. this is due to that we need to send $3\kappa + 4\rho$ bits per gate while they only need to send 2κ bits. On the other hand, our online communication is extremely small: it is about $4\times$ smaller than in the protocol of Nielsen et al. and 4–$6\times$ smaller than in the protocol of Rindal and Rosulek.

Acknowledgments

This material is based upon work supported by NSF awards #1111599 and #1563722; Samuel Ranellucci is also supported by NSF award #1564088.

References

[AMPR14] Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In *Advances in Cryptology—Eurocrypt 2014*, volume 8441 of *LNCS*, pages 387–404. Springer, 2014.

[BMR90] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *22nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 503–513. ACM Press, 1990.

[Bra13] Luís T. A. N. Brandão. Secure two-party computation with reusable bit-commitments, via a cut-and-choose with forge-and-lose technique. In *Advances in Cryptology—Asiacrypt 2013, Part II*, volume 8270 of *LNCS*, pages 441–463. Springer, 2013.

[CKMZ14] Seung Geol Choi, Jonathan Katz, Alex J. Malozemoff, and Vassilis Zikas. Efficient three-party computation from cut-and-choose. In *Advances in Cryptology—Crypto 2014, Part II*, volume 8617 of *LNCS*, pages 513–530. Springer, 2014.

[DI05] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In *Advances in Cryptology—Crypto 2005*, volume 3621 of *LNCS*, pages 378–394. Springer, 2005.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology—Crypto 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, 2012.

[FJN+13] Tore Kasper Frederiksen, Thomas Pelle Jakobsen, Jesper Buus Nielsen, Peter Sebastian Nordholt, and Claudio Orlandi. MiniLEGO: Efficient secure two-party computation from general assumptions. In *Advances in Cryptology—Eurocrypt 2013*, volume 7881 of *LNCS*, pages 537–556. Springer, 2013.

[FJN14] Tore Kasper Frederiksen, Thomas P. Jakobsen, and Jesper Buus Nielsen. Faster maliciously secure two-party computation using the GPU. In *9th Intl. Conf. on Security and Cryptography for Networks (SCN)*, volume 8642 of *LNCS*, pages 358–379. Springer, 2014.

[FJNT15] Tore Kasper Frederiksen, Thomas P. Jakobsen, Jesper Buus Nielsen, and Roberto Trifiletti. TinyLEGO: An interactive garbling scheme for maliciously secure two-party computation. Cryptology ePrint Archive, Report 2015/309, 2015. http://eprint.iacr.org/2015/309.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game, or a completeness theorem for protocols with honest majority. In *19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229. ACM Press, 1987.

[HKE13] Yan Huang, Jonathan Katz, and David Evans. Efficient secure two-party computation using symmetric cut-and-choose. In *Advances in Cryptology—Crypto 2013, Part II*, volume 8043 of *LNCS*, pages 18–35. Springer, 2013.

[HKK+14] Yan Huang, Jonathan Katz, Vladimir Kolesnikov, Ranjit Kumaresan, and Alex J. Malozemoff. Amortizing garbled circuits. In *Advances in Cryptology—Crypto 2014, Part II*, volume 8617 of *LNCS*, pages 458–475. Springer, 2014.

[HZ15] Yan Huang and Ruiyu Zhu. Revisiting LEGOs: Optimizations, analysis, and their limit. Cryptology ePrint Archive, Report 2015/1038, 2015. http://eprint.iacr.org/2015/1038.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer—efficiently. In *Advances in Cryptology—Crypto 2008*, volume 5157 of *LNCS*, pages 572–591. Springer, 2008.

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In *23rd ACM Conf. on Computer and Communications Security (CCS)*, pages 830–842. ACM Press, 2016.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *35th Intl. Colloquium on Automata, Languages, and Programming (ICALP), Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, 2008.

[Lin13] Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In *Advances in Cryptology—Crypto 2013, Part II*, volume 8043 of *LNCS*, pages 1–17. Springer, 2013.

[LOP11] Yehuda Lindell, Eli Oxman, and Benny Pinkas. The IPS compiler: Optimizations, variants and concrete efficiency. In *Advances in Cryptology—Crypto 2011*, volume 6841 of *LNCS*, pages 259–276. Springer, 2011.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Advances in Cryptology—Eurocrypt 2007*, volume 4515 of *LNCS*, pages 52–78. Springer, 2007.

[LP11] Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In *8th Theory of Cryptography Conference—TCC 2011*, volume 6597 of *LNCS*, pages 329–346. Springer, 2011. Available at `http://eprint.iacr.org/2010/284`.

[LPSY15] Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In *Advances in Cryptology—Crypto 2015, Part II*, volume 9216 of *LNCS*, pages 319–338. Springer, 2015.

[LR14] Yehuda Lindell and Ben Riva. Cut-and-choose Yao-based secure computation in the online/offline and batch settings. In *Advances in Cryptology—Crypto 2014, Part II*, volume 8617 of *LNCS*, pages 476–494. Springer, 2014.

[LR15] Yehuda Lindell and Ben Riva. Blazing fast 2PC in the offline/online setting with security for malicious adversaries. In *22nd ACM Conf. on Computer and Communications Security (CCS)*, pages 579–590. ACM Press, 2015.

[MOR16] Payman Mohassel, Ostap Orobets, and Ben Riva. Efficient server-aided 2PC for mobile phones. *Proc. Privacy Enhancing Technologies*, 2016(2):82–99, 2016.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *Advances in Cryptology—Crypto 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, 2012.

[NO09] Jesper Buus Nielsen and Claudio Orlandi. LEGO for two-party secure computation. In *6th Theory of Cryptography Conference—TCC 2009*, volume 5444 of *LNCS*, pages 368–386. Springer, 2009.

[NST17] Jesper Nielsen, Thomas Schneider, and Roberto Trifiletti. Constant-round maliciously secure 2PC with function-independent preprocessing using LEGO. In *Network and Distributed System Security Symposium (NDSS)*, 2017.

[RR16] Peter Rindal and Mike Rosulek. Faster malicious 2-party secure computation with online/offline dual execution. In *Proc. 25th USENIX Security Symposium*, pages 297–314. USENIX Association, 2016.

[SS11] Abhi Shelat and Chih-Hao Shen. Two-output secure computation with malicious adversaries. In *Advances in Cryptology—Eurocrypt 2011*, volume 6632 of *LNCS*, pages 386–405. Springer, 2011.

[WMK16] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. Faster two-party computation secure against malicious adversaries in the single-execution setting. Cryptology ePrint Archive, Report 2016/762, 2016. http://eprint.iacr.org/2016/762.

[Yao86] Andrew C.-C. Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 162–167. IEEE, 1986.