Global-Scale Secure Multiparty Computation

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

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

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

Abstract

We propose a new, constant-round protocol for multi-party computation of boolean circuits that is secure against an arbitrary number of malicious corruptions. At a high level, we extend and generalize recent work of Wang et al. in the two-party setting and design an efficient preprocessing phase that allows the parties to generate authenticated information; we then show how to use this information to distributively construct a *single* "authenticated" garbled circuit that is evaluated by one party.

Our resulting protocol improves upon the state-of-the-art both asymptotically and concretely. We validate these claims via several experiments demonstrating both the *efficiency* and *scalability* of our protocol:

- **Efficiency:** For three-party computation over a LAN, our protocol requires only 95 ms to evaluate AES. This is roughly a 700× improvement over the best prior work, and only 2.5× slower than the best known result in the *two*-party setting.

  In general, for n parties our protocol improves upon prior work (which was never implemented) by a factor of more than $230n$, e.g., an improvement of 3 orders of magnitude for 5-party computation.
- **Scalability:** We successfully executed our protocol with a large number of parties located all over the world, computing (for example) AES with 128 parties across 5 continents in under 3 minutes. Our work represents the largest-scale demonstration of secure computation to date.

1 Introduction

Secure multi-party computation (MPC) allows a set of parties to jointly perform a distributed computation while ensuring correctness, privacy of the parties' inputs, and more. MPC protocols can be classified in various ways depending on the native computations they support and the class of adversarial behavior they tolerate. With regard to the first of these, protocols are typically designed to compute either boolean circuits or arithmetic circuits over a large field. Although these models are equivalent in terms of their expressive power, arithmetic circuits for many natural computational tasks (e.g., comparisons, divisions, bit-wise operations, etc.) can be much larger than the corresponding boolean circuit for the same task, as well as more cumbersome to design.

With regard to security, some protocols tolerate only semi-honest adversaries that are assumed to follow the prescribed protocol but then try to learn additional information from the transcript of the execution. In contrast, the stronger malicious model does not make any assumptions about the behavior of the corrupted parties. Finally, some protocols are only secure for some threshold of corrupted parties. The standard options here are security assuming an honest majority (i.e., as long as strictly fewer than 1/2 of the parties are corrupted), or security for any number of corruptions (i.e., even if only of the parties is honest).

In this work we focus on MPC protocols tolerating any number of malicious corruptions. Most existing implementations of MPC protocols in this adversarial model rely on some variant of the secret-sharing paradigm introduced by Goldreich, Micali, and Wigderson [GMW87]. At a high level, this technique requires the parties to maintain the invariant of holding a linear secret sharing of the values on the wires of the circuit, along with some sort of authentication information on those shares. Linear gates in the circuit (e.g., XOR,

Paper	Comm./Comp. Complexity	Rounds
[BLN$^+$15]	$O\left(\lvert\mathcal{C}\rvert B^3 n\right)$	$O(d)$
[FKOS15]	$O\left(\lvert\mathcal{C}\rvert B^2 n\right)$	$O(d)$
[LPSY15] + [KOS16]	$O\left(\lvert\mathcal{C}\rvert \kappa n^2\right)$	$O(1)$
This paper	$O\left(\lvert\mathcal{C}\rvert Bn\right)$	$O(1)$

Table 1: Asymptotic complexity (per party) for n-party MPC protocols for boolean circuits, secure against an arbitrary number of malicious corruptions. Here, κ (resp., ρ) is the computational (resp., statistical) security parameter, $|C|$ is the circuit size, d is the depth of the circuit, and $B = O(\rho/\log|C|)$.

Setting	Setup	Indep.	Depen.	Online	Total
3PC-LAN	36	47	12	2	95
128PC-LAN	390	2727	11670	1870	16657
14PC-Worldwide	8711	9412	1947	250	20320
128PC-Worldwide	88056	30796	22659	2316	143827

Table 2: Selected performance results for our protocol. All results are in milliseconds. We consider the following settings (see Section 6 for more details.):
(a) **3PC-LAN**: three-party computation over a LAN;
(b) **128PC-LAN**: 128-party computation over a LAN;
(c) **14PC-Worldwide**: 14-party computation over a WAN, with parties located in 14 different cities across five continents;
(d) **128PC-Worldwide**: 128-party computation over a WAN, with parties located in 8 different cities across five continents (each city with 16 parties).

ADD) can be processed locally, while non-linear operations (e.g., AND, MULT) are handled by having the parties interact with each other to maintain the desired invariant. The most notable example of a protocol in this framework is perhaps SPDZ [DPSZ12, DKL$^+$13, KOS16], which supports arithmetic circuits; protocols for boolean circuits have also been designed [LOS14, FKOS15].

Although this approach can lead to protocols with reasonable efficiency when run over a LAN, it suffers the inherent drawback of leading to round complexity *linear* in the depth of the circuit being evaluated. This can have a significant impact on the overall efficiency when the parties running the protocol are geographically separated, or when the number of parties is high, and the communication latency dominates the cost of the protocol. For example, the communication latency between parties located in the U.S. and Europe is around 75 ms even with the dedicated network provided by Amazon EC2. If such parties are evaluating, say, SHA-256 (which has a circuit depth of about 4000), then a linear-round protocol requires $300,000$ ms just for the back-and-forth interaction between those parties, not even counting the time required for performing local cryptographic operations or transmitting any data.

Constant-round protocols tolerating any number of malicious corruptions have also been designed. The basic approach here, first proposed by Beaver, Micali, and Rogaway [BMR90], is to have the parties run a linear-round secure-computation protocol to compute a garbled circuit [Yao86] for the function f of interest; the parties can then evaluate that garbled circuit using a constant number of additional rounds. Since the circuit for computing the garbling of f has depth independent of f, the overall number of rounds is constant. Although several recent papers have explored this approach [CKMZ14, LPSY15, LSS16], these investigations have remained largely theoretical since the overall cost of even the best protocol using this approach is asymptotically worse than the nonconstant-round protocols mentioned above; see Table 1. In fact, prior implementations of constant-round MPC consider only the *semi-honest* setting [BDNP08, BLO16].

1.1 Our Contributions

In this paper, we take a significant step towards practical MPC tolerating an unbounded number of malicious corruptions. To this end, we propose a new, constant-round protocol for multi-party computation of boolean circuits secure in this setting. Our protocol extends and generalizes the recent work of Wang et al. [WRK17] in the two-party setting. Specifically, we design an optimized, multiparty version of their TinyOT protocol so as to enable n parties to generate certain authenticated information as part of a preprocessing phase. Next, generalizing their main protocol, we show how to use this information to distributively construct a *single* "authenticated" garbled circuit that is evaluated by a single party. (An overview of our entire protocol appears in Section 3, with details in the remainder of the paper.)

Our protocol improves upon the state-of-the-art both asymptotically (cf. Table 1) and concretely (see Section 6.4), and in particular it allows us to give the *first* implementation of a constant-round MPC protocol with malicious security. Our experiments demonstrate that our protocol is both *efficient* and *scalable*:

- **Efficiency:** For three-party computation over a LAN, our protocol requires only 95 ms to securely evaluate AES. This is roughly a 700× improvement over the best prior work, and only 2.5× slower than the best known result in the *two*-party setting. In general, for n parties our protocol improves upon the best prior work [LPSY15, LSS16] (which was not implemented) by a factor of more than $200n$.
- **Scalability:** We successfully executed our protocol with many parties located all over the world, computing (for example) AES with 128 parties across 5 continents in under 3 minutes. To the best of our knowledge, our work represents the largest-scale demonstration of secure computation to date, even considering weaker adversarial models.

Selected performance results for our protocol are reported in Table 2. Following [NST17], in the table we divide execution into different phases:

- **Setup.** Here, parties do not know the number of executions or the circuit size. For example, baseOT can be performed in this phase.
- **Function-independent preprocessing.** Here, the parties need not know their inputs or the function to compute (besides an upper bound on the number of gates).
- **Function-dependent preprocessing.** The parties know the function being computed, but need not know their inputs.
- **Online phase.** The parties evaluate the function on their inputs.

1.2 Related Work

Implementations of MPC protocols. The first implementations of generic MPC assumed a semi-honest adversary corrupting a minority of the parties. Early work in this area includes FairplayMP [BDNP08] for boolean circuits, and VIFF [DGKN09] and SEPIA [BSMD10] for arithmetic circuits. Implementations of protocols handling an arbitrary number of corrupted parties, but still in the semi-honest setting, were shown by Choi et al. [CHK$^+$12] and Ben-Efraim et al. [BLO16], the latter running in a constant number of rounds.

There are fewer implementations of MPC protocols handling *malicious* attackers. Jakobsen et al. [JMN10] developed the first such system. SPDZ and its subsequent improvements [DPSZ12, DKL$^+$13, KSS13, KOS16] greatly improved the efficiency. As noted earlier, all existing implementations of MPC tolerating malicious attackers have round complexity linear in the dept of the circuit.

Another line of work has specifically targeted *three*-party computation. Implementations here include Sharemind [BLW08], the sugar-beet auction run by Bogetoft et al. [BCD$^+$09], and the recent work of Araki et al. [AFL$^+$16]; these each tolerate only semi-honest behavior. Mohassel et al. [MRZ15] and Furukawa et al. [FLNW17] tolerate a malicious attacker corrupting only one party.

Constant-round MPC. Techniques for building constant-round MPC protocols have been studied in various settings. As noted above, FairplayMP [BDNP08] and Ben-Efraim et al. [BLO16] implemented this approach in the semi-honest setting. Other researchers have proposed approaches without providing an implementation, possibly because an implementation would be too complex or because the concrete efficiency of the resulting protocol would be uncompetitive with nonconstant-round protocols. As examples, Damgård and Ishai [DI05] proposed a protocol making black-box use of the underlying cryptographic primitives, and Choi et al. [CKMZ14] looked at the three-party setting with malicious corruption of any number of parties. Lindell et al. [LPSY15, LSS16] considered optimizations of the BMR approach in the malicious setting. We compare the efficiency of our protocol with relevant prior work in Section 6.4.

Concurrent work. Concurrently and independently of our work, Hazay et al. [HSSV17] proposed an MPC protocol running in two stages that achieves similar asymptotic complexity as ours. We note several differences between our work and theirs: (1) our preprocessing protocol for generating authenticated information is more efficient than theirs both asymptotically and concretely (see Section 6.4); (2) our main protocol (i.e., generating a single authenticated garbled circuit using the information from the preprocessing phase) is very different from theirs; (3) they currently offer no implementation of their protocol.

2 Notation and Preliminaries

We use κ and ρ to denote the computational and statistical security parameters, respectively. We also use $=$ to denote equality and $:=$ to denote assignment.

A circuit is represented as a list of gates of the form $(\alpha, \beta, \gamma, T)$, where this represents a gate with input wires α and β, output wire γ, and gate type $T \in \{\oplus, \wedge\}$. Parties are denoted by $P_1, \ldots, P_n$. We use $\mathcal{I}_i$ to denote the set of all input-wire indices for P_i's input, $\mathcal{W}$ to denote the set of output-wire indices for all AND gates, and $\mathcal{O}$ to denote the set of output-wire indices of the circuit. $\mathcal{M}$ is used to denote the set of all corrupted parties, with $\mathcal{H} = [n] \setminus \mathcal{M}$ denoting the set of all honest parties.

Our protocol operates by having the parties distributively construct a garbled circuit that is evaluated by one of the parties; we let P_1 be the circuit evaluator.

Authenticated bits. Information-theoretic message authentication codes (IT-MACs) for authenticating bits were first used for efficient secure computation by Nielsen et al. [NNOB12] in the two-party setting. The idea can be extended to the multiparty setting as follows. Each player holds a global MAC key $\Delta_i \in \{0,1\}^\kappa$. When P_i holds a bit x authenticated by P_j, this means that P_j is given a random key $\mathsf{K}_j[x] \in \{0,1\}^\kappa$ and P_i is given the MAC tag $\mathsf{M}_j[x] := \mathsf{K}_j[x] \oplus x\Delta_j$. We let $[x]^i$ denote an authenticated bit where the value of x is known to P_i, and is authenticated to all other parties. In more detail, $[x]^i$ means that $(x, \{\mathsf{M}_k[x]\}_{k \neq i})$ is given to P_i, and $\mathsf{K}_j[x]$ is given to P_j for $j \neq i$.

Note that $[x]^i$ is XOR-homomorphic: given two authenticated bits $[x]^i, [y]^i$ known to the same party P_i, it is possible to locally compute the authenticated bit $[z]^i$ with $z = x \oplus y$ as follows:

- P_i computes $z := x \oplus y$, and $\{\mathsf{M}_j[z] := \mathsf{M}_j[x] \oplus \mathsf{M}_j[y]\}_{j \neq i}$;
- P_j (for $j \neq i$) computes $\mathsf{K}_j[z] := \mathsf{K}_j[x] \oplus \mathsf{K}_j[y]$.

Parties can also locally negate $[x]^i$, resulting in $[z]^i$ with $z = \bar{x}$:

- P_i computes $z := x \oplus 1$ and $\{\mathsf{M}_j[z] := \mathsf{M}_j[x]\}_{j \neq i}$;
- P_j (for $j \neq i$) computes $\mathsf{K}_j[z] := \mathsf{K}_j[x] \oplus \Delta_j$.

We let $\mathcal{F}^n_{\mathsf{aBit}}$ denote an ideal functionality that distributes authenticated bits to the parties.(Details see Figure 4 in Section 5.1). We also discuss an efficient instantiation of this functionality in Section 5.1.

Note that the above representation assumes that P_i uses a single global MAC key Δ_i. In cases where other keys are used, we explicitly add a subscript to the representation, i.e., we use $\mathsf{K}_i[x]_{G_i}$ and $\mathsf{M}_i[x]_{G_i} = \mathsf{K}_i[x]_{G_i} \oplus xG_i$ to denote the key and MAC tag in this case.

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

Honest parties:

1. Upon receiving init from all parties, sample $\{\Delta_i \in \{0,1\}^\kappa\}_{i\in[n]}$ and send Δ_i to P_i.
2. Upon receiving random from all P_i, sample a random bit r and a random authenticated share $\langle r\rangle = \{(r^i, \{\mathsf{M}_j[r^i], \mathsf{K}_i[r^j]\}_{j\neq i})\}_{i\in[n]}$. For each $i \in [n]$, the box sends $(r^i, \{\mathsf{M}_j[r^i], \mathsf{K}_i[r^j]\}_{j\neq i})$ to P_i.
3. Upon receiving $\left(\mathtt{AND}, (r^i, \{\mathsf{M}_j[r^i], \mathsf{K}_i[r^j]\}_{j\neq i}), (s^i, \{\mathsf{M}_j[s^i], \mathsf{K}_i[s^j]\}_{j\neq i}\right)$ from P_i for all $i \in [n]$, the box checks that all MACs are valid, computes $t := \left(\bigoplus_{i\in[n]} r^i\right) \wedge \left(\bigoplus_{i\in[n]} s^i\right)$ and picks a random authenticated share $\langle t\rangle = \{(t^i, \{\mathsf{M}_j[t^i], \mathsf{K}_i[t^j]\}_{j\neq i})\}_{i\in[n]}$. For each $i \in [n]$, the box sends $(t^i, \{\mathsf{M}_j[t^i], \mathsf{K}_i[t^j]\}_{j\neq i})$ to P_i.

Corrupted parties: Corrupted parties can choose randomness used to compute the value they receive from the functionality.

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

Figure 1: The multi-party preprocessing functionality.

Authenticated shares. In the above construction, x is known to one party. To generate an authenticated *shared* bit x, where x is not known to any party, we generate XOR-shares for x (i.e., shares $\{x^i\}$ with $\bigoplus_i x^i = x$) and then distribute the authenticated bits $\{[x^i]^i\}$. We let $\langle x\rangle$ denote the collection of these authenticated shares for x; that is, $\langle x\rangle$ means that each party P_i holds $\left(x^i, \{\mathsf{M}_j[x^i], \mathsf{K}_i[x^j]\}_{j\neq i}\right)$.

We let $\mathcal{F}_{\mathsf{aShare}}$ denote an ideal functionality that distributes authenticated shared bits to the parties (Details see Figure 5 in Section 5.2). We discuss an efficient instantiation of this functionality in Section 5.2.

Definition of security. We use the standard notion of (standalone) security against an unbounded number of malicious parties. As our protocol is described, only P_1 receives output; however, it would not be difficult to modify the protocol (using standard techniques) to support arbitrary outputs for different parties. In that case our protocol would achieve the notion of secure computation with designated abort [GL05, Gol09].

3 Overview of Our Main Protocol

Our main protocol is designed in the $\mathcal{F}_{\mathsf{Pre}}$-hybrid model (see Figure 1). At a high level, $\mathcal{F}_{\mathsf{Pre}}$ generates authenticated shares on random bits x, y, z such that $z = x \wedge y$. (We refer to these as *AND triples.*) Our main protocol then uses those authenticated shares to distributively construct a single, "authenticated" garbled circuit that is evaluated by one of the parties. In the remainder of this section, we describe our main protocol; further details are in Section 4. In Section 5 we then discuss how to efficiently realize $\mathcal{F}_{\mathsf{Pre}}$.

3.1 Two-Party Authenticated Garbling

Wang et al. [WRK17] recently proposed a maliciously secure protocol for *two*-party computation using a two-party version of $\mathcal{F}_{\mathsf{Pre}}$. Before explaining how we extend it to the multiparty setting, we briefly describe their protocol partially based on their presentation.

Their protocol is based on a standard garbled circuit, where each wire α is associated with a random "mask" $\lambda_\alpha \in \{0,1\}$ known to $\mathsf{P_A}$. If the actual wire value when the circuit is evaluated is x, then the masked value observed by the circuit evaluator (namely, $\mathsf{P_B}$) will be $\hat{x} = x \oplus \lambda_\alpha$. 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 $\mathsf{P_A}$. $\mathsf{P_B}$ also learns $\mathsf{L}_{\alpha,\hat{x}}$ for that wire. Let H be a hash function modeled as a random oracle. The garbled table for an AND gate $(\alpha, \beta, \gamma, \wedge)$ is given by:

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

Table 3: Final construction of an authenticated garbled table for an AND gate.

$\hat{x}$ $\hat{y}$	truth table	garbled table
0 0	$\hat{z}_{00} = (\lambda_\alpha \wedge \lambda_\beta) \oplus \lambda_\gamma$	$H(\mathsf{L}_{\alpha,0}, \mathsf{L}_{\beta,0}, \gamma, 00) \oplus (\hat{z}_{00}, \mathsf{L}_{\gamma,\hat{z}_{00}})$
0 1	$\hat{z}_{01} = (\lambda_\alpha \wedge \overline{\lambda_\beta}) \oplus \lambda_\gamma$	$H(\mathsf{L}_{\alpha,0}, \mathsf{L}_{\beta,1}, \gamma, 01) \oplus (\hat{z}_{01}, \mathsf{L}_{\gamma,\hat{z}_{01}})$
1 0	$\hat{z}_{10} = (\overline{\lambda_\alpha} \wedge \lambda_\beta) \oplus \lambda_\gamma$	$H(\mathsf{L}_{\alpha,1}, \mathsf{L}_{\beta,0}, \gamma, 10) \oplus (\hat{z}_{10}, \mathsf{L}_{\gamma,\hat{z}_{10}})$
1 1	$\hat{z}_{11} = (\overline{\lambda_\alpha} \wedge \overline{\lambda_\beta}) \oplus \lambda_\gamma$	$H(\mathsf{L}_{\alpha,1}, \mathsf{L}_{\beta,1}, \gamma, 11) \oplus (\hat{z}_{11}, \mathsf{L}_{\gamma,\hat{z}_{11}})$

$\mathsf{P_B}$, holding $(\hat{x}, \mathsf{L}_{\alpha,\hat{x}})$ and $(\hat{y}, \mathsf{L}_{\beta,\hat{y}})$, evaluates it by picking the $(\hat{x}, \hat{y})$-th row and decrypting using the garbled labels it holds, thus obtaining $(\hat{z}, \mathsf{L}_{\gamma,\hat{z}})$. This is not secure because a malicious party can perform a selective failure attack to learn, e.g., $\hat{x}$. However, Wang et al. observe that it can be prevented if two parties hold *secret shares* of the garbled table: e.g., for the first row, $\mathsf{P_A}$ knows $(r_{00}, \mathsf{L}^\mathsf{A}_{\gamma,\hat{z}_{00}})$, while $\mathsf{P_B}$ knows $(s_{00} = \hat{z}_{00} \oplus r_{00}, \mathsf{L}^\mathsf{B}_{\gamma,\hat{z}_{00}})$, where $\mathsf{L}_{\gamma,z} = \mathsf{L}^\mathsf{A}_{\gamma,z} \oplus \mathsf{L}^\mathsf{B}_{\gamma,z}$. Once $\mathsf{P_A}$ sends its shares of all the garbled gates, $\mathsf{P_B}$ can XOR those shares with its own and then evaluate the garbled circuit as before.

Informally, it ensures *privacy* against a malicious $\mathsf{P_A}$ since the results of any changes $\mathsf{P_A}$ makes to the garbled circuit are *independent* of $\mathsf{P_B}$'s inputs. However, $\mathsf{P_A}$ can still affect *correctness* by, e.g., flipping the masked value in a row. This is solved using information-theoretic MACs on $\mathsf{P_A}$'s share of masked bits. The shares of the garbled table now take the form as shown in the table below.

$\hat{x}$ $\hat{y}$	$\mathsf{P_A}$'s share of garbled table	$\mathsf{P_B}$'s share of garbled table
0 0	$H(\mathsf{L}_{\alpha,0}, \mathsf{L}_{\beta,0}, \gamma, 00) \oplus (r_{00}, \mathsf{M}[r_{00}], \mathsf{L}^\mathsf{A}_{\gamma,\hat{z}_{00}})$	$(s_{00} = \hat{z}_{00} \oplus r_{00}, \mathsf{K}[r_{00}], \mathsf{L}^\mathsf{B}_{\gamma,\hat{z}_{00}})$
0 1	$H(\mathsf{L}_{\alpha,0}, \mathsf{L}_{\beta,1}, \gamma, 01) \oplus (r_{01}, \mathsf{M}[r_{01}], \mathsf{L}^\mathsf{A}_{\gamma,\hat{z}_{01}})$	$(s_{01} = \hat{z}_{01} \oplus r_{01}, \mathsf{K}[r_{01}], \mathsf{L}^\mathsf{B}_{\gamma,\hat{z}_{01}})$
1 0	$H(\mathsf{L}_{\alpha,1}, \mathsf{L}_{\beta,0}, \gamma, 10) \oplus (r_{10}, \mathsf{M}[r_{10}], \mathsf{L}^\mathsf{A}_{\gamma,\hat{z}_{10}})$	$(s_{10} = \hat{z}_{10} \oplus r_{10}, \mathsf{K}[r_{10}], \mathsf{L}^\mathsf{B}_{\gamma,\hat{z}_{10}})$
1 1	$H(\mathsf{L}_{\alpha,1}, \mathsf{L}_{\beta,1}, \gamma, 11) \oplus (r_{11}, \mathsf{M}[r_{11}], \mathsf{L}^\mathsf{A}_{\gamma,\hat{z}_{11}})$	$(s_{11} = \hat{z}_{11} \oplus r_{11}, \mathsf{K}[r_{11}], \mathsf{L}^\mathsf{B}_{\gamma,\hat{z}_{11}})$

$\mathsf{P_B}$ 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, which will occur independent of $\mathsf{P_B}$'s actual input.

Finally they observe that if setting $\Delta = \Delta_\mathsf{A}$ then $\mathsf{L}_{\gamma,\hat{z}_{00}}$ can be efficiently secret-shared:

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

Recall that $\mathsf{P_A}$ knows $L_{\gamma,0}$, r_{00} and Δ_A. Their key insight is that if s_{00} is an authenticated bit known to $\mathsf{P_B}$, then $\mathsf{P_A}$ can locally compute the share $\mathsf{L}^\mathsf{A}_{\gamma,\hat{z}_{00}} := \mathsf{L}_{\gamma,0} \oplus r_{00}\Delta_\mathsf{A} \oplus \mathsf{K}[s_{00}]$ from the information it has, and then the other share $\mathsf{L}^\mathsf{B}_{\gamma,\hat{z}_{00}} := \mathsf{K}[s_{00}] \oplus s_{00}\Delta_\mathsf{A}$ is just the MAC on s_{00} that $\mathsf{P_B}$ already holds! Table 3 shows the garbled table based on this observation.

3.2 Extension the Multiparty Setting

It is not trivial to extend the above protocol to the multiparty setting. The main challenge is that even when $n-1$ parties are corrupted, we still need to make sure that the adversary cannot learn any information about the honest party's inputs.

Attempted ideas. One idea, adopted by Choi et al. [CKMZ14] in the three-party setting, is to let $n-1$ parties jointly compute a garbled circuit that the remaining party will evaluate. However, if the $n-1$ garblers are corrupt, there is no guarantee about the correctness of the garbled circuit they generate. For that reason, Choi et al. had to use cut-and-choose to check correctness of a random subset of ρ garbled circuits, which imposes a huge overhead.

To avoid this additional cut-and-choose, we would like all parties to be involved in the garbled-circuit generation, as in the BMR protocol [BMR90]. However, state-of-the-art protocols based on BMR that are maliciously secure against corruption of $n-1$ parties either require either $O(n)$ somewhat homomorphic encryptions [LSS16] or $O(n)$ SPDZ multiplication subprotocols [LPSY15] per AND gate both of which are relatively inefficient. We aim instead to use "simpler" TinyOT-like functionalities as we explain next.

Multiparty TinyOT: BDOZ-style vs. SPDZ-style. We observe that in the existing literature, there are mainly two flavors on how authenticated shared are constructed.

- **BDOZ-style** [BDOZ11]: For a secret bit x, each party holds a share of x. For each ordered pair of parties (P_i, P_j), P_i authenticates its own share (namely x^i) to P_j.
- **SPDZ-style** [DPSZ12]: Each party holds a share of a global MAC key. For a secret bit x, each party holds a share of x and a share of the MAC on x.

Note that these protocols are constructed for arithmetic circuits, but these representations also apply to binary circuits. Existing papers prefer SPDZ-style shares to BDOZ-style shares, because SPDZ-style shares is smaller and thus is more efficient to operate on. Indeed, existing papers that investigated protocols for multi-party TinyOT are all based on SPDZ-style shares [LOS14, BLN$^+$15, FKOS15].

Our key observation is that such SPDZ-style AND triple, although efficient for interactive MPC protocols, is not suitable for our use to construct constant-round MPC protocols. In particular, in the SPDZ-style shares, each parties knows Δ_i as a share of the global key $\Delta = \bigoplus_i \Delta_i$. For each bit x, they holds shares of $x\Delta$. Since Δ is not known to any party, it is not directly related to any garbled circuit. On the contrary, in the BDOZ-style protocols, each party holds $(x^i, \{\mathsf{M}_j[x^i], \mathsf{K}_i[x^j]\}_{j\neq i})$, as we have already described in Section 2. In this case, they essentially hold shares of $x\Delta_i$ for all $i \in [n]$, because:

$$\begin{aligned} x\Delta_i &= \left(\bigoplus_j x^j\right)\Delta_i = x^i\Delta_i \oplus \left(\bigoplus_{j\neq i} x^j\Delta_i\right) \\ &= x^i\Delta_i \oplus \left(\bigoplus_{j\neq i} \mathsf{M}_i[x^j] \oplus \mathsf{K}_i[x^j]\right) \\ &= \left(x^i\Delta_i \oplus \bigoplus_{j\neq i} \mathsf{K}_i[x^j]\right) \oplus \bigoplus_{j\neq i} \mathsf{M}_i[x^j] \end{aligned}$$

Here, P_i knows the first value, while each P_j with $j \neq i$ knows $\mathsf{M}_i[x^j]$. In other word, a BDOZ-style share of a bit x can be used to construct shares of $x\Delta_i$ for each i. This can further be used to construct shares of garbled labels, *if we use the same Δ_i for authenticated shares and the global difference used in free-XOR*! Indeed, looking ahead to the main protocol in Figure 2 step 4 (d), the content of the garbled circuit can be viewed as some authentication information plus shares of the garbled output labels for each garbler!

High level picture of the protocol. Given the above discussion, we can now picture the high level idea of our protocol. Our idea, from a high level view, is to let $n-1$ parties be garblers, each maintaining a set of garbled labels, and to let the remaining party be the evaluator. Each garbler knows its own set of garbled labels. However, for each gate and for each garbler, the *permuted garbled output labels* are secretly shared to all parties, and thus no party knows how these labels are permuted. For each garbler P_i and a garbler row, P_i has shares of permuted garbled output labels for all garblers. In the garbled table, P_i encrypts all these shares using its own set of garbled input labels. Further, shares of the mask value are authenticated

Protocol Π_{mpc}

Inputs: In the function-independent phase, parties know $|\mathcal{C}|$ and $|\mathcal{I}|$; in the function-dependent phase, parties get a circuit representing function $f : \{0,1\}^{|\mathcal{I}_1|} \times ... \times \{0,1\}^{|\mathcal{I}_n|} \to \{0,1\}^{|\mathcal{O}|}$; in the input-processing phase, P_i holds $x_i \in \{0,1\}^{|\mathcal{I}_i|}$.

Function-independent phase:

1. P_i sends init to $\mathcal{F}_{\mathsf{Pre}}$, which sends Δ_i to P_i.
2. For each wire $w \in \mathcal{I} \cup \mathcal{W}, i \in [n]$, P_i sends random to $\mathcal{F}_{\mathsf{Pre}}$, which sends $\left(r_w^i, \{\mathsf{M}_j[r_w^i], \mathsf{K}_i[r_w^j]\}_{j \neq i}\right)$ to P_i, where $\bigoplus_{i \in [n]} r_w^i = \lambda_w$. For each $i \neq 1$, P_i also picks a random κ-bit string $\mathsf{L}_{w,0}^i$.

Function-dependent phase:

3. For each gate $\mathcal{G} = (\alpha, \beta, \gamma, \oplus)$, each $i \in [n]$, P_i computes $\left(r_\gamma^i, \{\mathsf{M}_j[r_\gamma^i], \mathsf{K}_i[r_\gamma^j]\}_{j \neq i}\right) := \left(r_\alpha^i \oplus r_\beta^i, \left\{\mathsf{M}_j[r_\alpha^i] \oplus \mathsf{M}_j[r_\beta^i], \mathsf{K}_i[r_\alpha^j] \oplus \mathsf{K}_i[r_\beta^j]\right\}_{j \neq i}\right)$. For each $i \neq 1$, P_i also computes $\mathsf{L}_{\gamma,0}^i := \mathsf{L}_{\alpha,0}^i \oplus \mathsf{L}_{\beta,0}^i$.
4. For each gate $\mathcal{G} = (\alpha, \beta, \gamma, \wedge)$:

   (a) For each $i \in [n]$, P_i sends $\left(\mathsf{and}, \left(r_\alpha^i, \{\mathsf{M}_j[r_\alpha^i], \mathsf{K}_i[r_\alpha^j]\}_{j \neq i}\right), \left(r_\beta^i, \left\{\mathsf{M}_j[r_\beta^i], \mathsf{K}_i[r_\beta^j]\right\}_{j \neq i}\right)\right)$ to $\mathcal{F}_{\mathsf{Pre}}$, which sends $\left(r_\sigma^i, \{\mathsf{M}_j[r_\sigma^i], \mathsf{K}_i[r_\sigma^j]\}_{j \neq i}\right)$ to P_i, where $\bigoplus_{i \in [n]} r_\sigma^i = \left(\bigoplus_{i \in [n]} r_\alpha^i\right) \wedge \left(\bigoplus_{i \in [n]} r_\beta^i\right)$.

   (b) For each $i \neq 1$, P_i computes the following locally.
$$\begin{aligned}
\left(r_{\gamma,0}^i, \{\mathsf{M}_j[r_{\gamma,0}^i], \mathsf{K}_i[r_{\gamma,0}^j]\}_{j \neq i}\right) &:= \left(r_\sigma^i \oplus r_\gamma^i, \left\{\mathsf{M}_j[r_\sigma^i] \oplus \mathsf{M}_j[r_\gamma^i], \quad \mathsf{K}_i[r_\sigma^j] \oplus \mathsf{K}_i[r_\gamma^j]\right\}_{j \neq i}\right) \\
\left(r_{\gamma,1}^i, \{\mathsf{M}_j[r_{\gamma,1}^i], \mathsf{K}_i[r_{\gamma,1}^j]\}_{j \neq i}\right) &:= \left(r_{\gamma,0}^i \oplus r_\alpha^i, \left\{\mathsf{M}_j[r_{\gamma,0}^i] \oplus \mathsf{M}_j[r_\alpha^i], \quad \mathsf{K}_i[r_{\gamma,0}^j] \oplus \mathsf{K}_i[r_\alpha^j]\right\}_{j \neq i}\right) \\
\left(r_{\gamma,2}^i, \{\mathsf{M}_j[r_{\gamma,2}^i], \mathsf{K}_i[r_{\gamma,2}^j]\}_{j \neq i}\right) &:= \left(r_{\gamma,0}^i \oplus r_\beta^i, \left\{\mathsf{M}_j[r_{\gamma,0}^i] \oplus \mathsf{M}_j[r_\beta^i], \quad \mathsf{K}_i[r_{\gamma,0}^j] \oplus \mathsf{K}_i[r_\beta^j]\right\}_{j \neq i}\right) \\
\left(r_{\gamma,3}^i, \{\mathsf{M}_j[r_{\gamma,3}^i], \mathsf{K}_i[r_{\gamma,3}^j]\}_{j \neq i}\right) &:= \Big(r_{\gamma,1}^i \oplus r_\beta^i, \left\{\mathsf{M}_1[r_{\gamma,1}^i] \oplus \mathsf{M}_1[r_\beta^i], \quad \mathsf{K}_i[r_{\gamma,1}^1] \oplus \mathsf{K}_i[r_\beta^1] \oplus \Delta_i\right\} \\
&\qquad \bigcup \left\{\mathsf{M}_j[r_{\gamma,1}^i] \oplus \mathsf{M}_j[r_\beta^i], \mathsf{K}_i[r_{\gamma,1}^j] \oplus \mathsf{K}_i[r_\beta^j]\right\}_{j \neq i,1}\Big)
\end{aligned}$$

   (c) P_1 computes the following locally.
$$\begin{aligned}
\left(r_{\gamma,0}^1, \{\mathsf{M}_j[r_{\gamma,0}^1], \mathsf{K}_1[r_{\gamma,0}^j]\}_{j \neq i}\right) &:= \left(r_\sigma^1 \oplus r_\gamma^1, \quad \left\{\mathsf{M}_j[r_\sigma^1] \oplus \mathsf{M}_j[r_\gamma^1], \quad \mathsf{K}_1[r_\sigma^j] \oplus \mathsf{K}_1[r_\gamma^j]\right\}_{j \neq i}\right) \\
\left(r_{\gamma,1}^1, \{\mathsf{M}_j[r_{\gamma,1}^1], \mathsf{K}_1[r_{\gamma,1}^j]\}_{j \neq i}\right) &:= \left(r_{\gamma,0}^1 \oplus r_\alpha^1, \quad \left\{\mathsf{M}_j[r_{\gamma,0}^1] \oplus \mathsf{M}_j[r_\alpha^1], \mathsf{K}_1[r_{\gamma,0}^j] \oplus \mathsf{K}_1[r_\alpha^j]\right\}_{j \neq i}\right) \\
\left(r_{\gamma,2}^1, \{\mathsf{M}_j[r_{\gamma,2}^1], \mathsf{K}_1[r_{\gamma,2}^j]\}_{j \neq i}\right) &:= \left(r_{\gamma,0}^1 \oplus r_\beta^1, \quad \left\{\mathsf{M}_j[r_{\gamma,0}^1] \oplus \mathsf{M}_j[r_\beta^1], \mathsf{K}_1[r_{\gamma,0}^j] \oplus \mathsf{K}_1[r_\beta^j]\right\}_{j \neq i}\right) \\
\left(r_{\gamma,3}^1, \{\mathsf{M}_j[r_{\gamma,3}^1], \mathsf{K}_1[r_{\gamma,3}^j]\}_{j \neq i}\right) &:= \left(r_{\gamma,1}^1 \oplus r_\beta^1 \oplus 1, \left\{\mathsf{M}_j[r_{\gamma,1}^1] \oplus \mathsf{M}_j[r_\beta^1], \mathsf{K}_1[r_{\gamma,1}^j] \oplus \mathsf{K}_1[r_\beta^j]\right\}_{j \neq i}\right)
\end{aligned}$$

   (d) For each $i \neq 1$, P_i computes $\mathsf{L}_{\alpha,1}^i := \mathsf{L}_{\alpha,0}^i \oplus \Delta_i$ and $\mathsf{L}_{\beta,1}^i := \mathsf{L}_{\beta,0}^i \oplus \Delta_i$, and sends the following to P_1.
$$\begin{aligned}
G_{\gamma,0}^i &:= H\left(\mathsf{L}_{\alpha,0}^i, \mathsf{L}_{\beta,0}^i, \gamma, 0\right) \oplus \left(r_{\gamma,0}^i, \{\mathsf{M}_j[r_{\gamma,0}^i]\}_{j \neq i}, \mathsf{L}_{\gamma,0}^i \oplus \left(\bigoplus_{j \neq i} \mathsf{K}_i[r_{\gamma,0}^j]\right) \oplus r_{\gamma,0}^i \Delta_i\right) \\
G_{\gamma,1}^i &:= H\left(\mathsf{L}_{\alpha,0}^i, \mathsf{L}_{\beta,1}^i, \gamma, 1\right) \oplus \left(r_{\gamma,1}^i, \{\mathsf{M}_j[r_{\gamma,1}^i]\}_{j \neq i}, \mathsf{L}_{\gamma,0}^i \oplus \left(\bigoplus_{j \neq i} \mathsf{K}_i[r_{\gamma,1}^j]\right) \oplus r_{\gamma,1}^i \Delta_i\right) \\
G_{\gamma,2}^i &:= H\left(\mathsf{L}_{\alpha,1}^i, \mathsf{L}_{\beta,0}^i, \gamma, 2\right) \oplus \left(r_{\gamma,2}^i, \{\mathsf{M}_j[r_{\gamma,2}^i]\}_{j \neq i}, \mathsf{L}_{\gamma,0}^i \oplus \left(\bigoplus_{j \neq i} \mathsf{K}_i[r_{\gamma,2}^j]\right) \oplus r_{\gamma,2}^i \Delta_i\right) \\
G_{\gamma,3}^i &:= H\left(\mathsf{L}_{\alpha,1}^i, \mathsf{L}_{\beta,1}^i, \gamma, 3\right) \oplus \left(r_{\gamma,3}^i, \{\mathsf{M}_j[r_{\gamma,3}^i]\}_{j \neq i}, \mathsf{L}_{\gamma,0}^i \oplus \left(\bigoplus_{j \neq i} \mathsf{K}_i[r_{\gamma,3}^j]\right) \oplus r_{\gamma,3}^i \Delta_i\right)
\end{aligned}$$

Figure 2: Our main protocol.

similarly. The evaluator decrypts the same row of garbled tables from all garblers in order to recompute the garbled output labels for each garbler. Intuitively, this ensures that for any set of $n-1$ parties, they cannot garble or evaluate any gate.

Protocol Π_{mpc}, continued

Input Processing:

5. For each $i \neq 1, w \in \mathcal{I}_i$, for each $j \neq i$, P_j sends $(r_w^j, \mathsf{M}_i[r_w^j])$ to P_i, who checks that $(r_w^j, \mathsf{M}_i[r_w^j], \mathsf{K}_i[r_w^j])$ is valid, and computes $x_w^i \oplus \lambda_w := x_w^i \oplus \left(\bigoplus_{i\in[n]} r_w^i\right)$. P_i broadcasts the value $x_w^i \oplus \lambda_w$. For each $j \neq 1$, P_j sends $\mathsf{L}^j_{x^i\oplus\lambda_w}$ to P_1.
6. For each $w \in \mathcal{I}_1, i \neq 1$, P_i sends $(r_w^i, \mathsf{M}_1[r_w^i])$ to P_1, who checks that $(r_w^i, \mathsf{M}_1[r_w^i], \mathsf{K}_1[r_w^i])$ are valid, and computes $x_w^1 \oplus \lambda_w := x_w^1 \oplus \left(\bigoplus_{i\in[n]} r_w^i\right)$. P_1 sends $x_w^1 \oplus \lambda_w$ to P_i, who sends $\mathsf{L}^i_{w,x_w^1\oplus\lambda_w}$ to P_1.

Circuit Evaluation:

7. P_1 evaluates the circuit following the topological order. For each gate $\mathcal{G} = (\alpha, \beta, \gamma, T)$, P_1 holds $\left(z_\alpha \oplus \lambda_\alpha, \{\mathsf{L}^i_{\alpha,z_\alpha\oplus\lambda_\alpha}\}_{i\neq 1}\right)$ and $\left(z_\beta \oplus \lambda_\beta, \left\{\mathsf{L}^i_{\beta,z_\beta\oplus\lambda_\beta}\right\}_{i\neq 1}\right)$, where z_α, z_β are the underlying values of the wire.
   (a) If $T = \oplus$, P_1 computes $z_\gamma \oplus \lambda_\gamma := (z_\alpha \oplus \lambda_\alpha) \oplus (z_\beta \oplus \lambda_\beta)$ and $\left\{\mathsf{L}^i_{\gamma,z_\gamma\oplus\lambda_\gamma} := \mathsf{L}^i_{\alpha,z_\alpha\oplus\lambda_\alpha} \oplus \mathsf{L}^i_{\beta,z_\beta\oplus\lambda_\beta}\right\}_{i\neq 1}$
   (b) If $T = \wedge$, P_1 computes $\ell := 2(z_\alpha \oplus \lambda_\alpha) + (z_\beta \oplus \lambda_\beta)$. For $i \neq 1$, P_1 computes
   $$\left(r^i_{\gamma,\ell}, \left\{\mathsf{M}_j[r^i_{\gamma,\ell}]\right\}_{j\neq i}, \mathsf{L}^i_\gamma\right) := G^i_{\gamma,\ell} \oplus H\left(\mathsf{L}^i_{\alpha,z_\alpha\oplus\lambda_\alpha}, \mathsf{L}^i_{\beta,z_\beta\oplus\lambda_\beta}, \gamma, \ell\right).$$
   P_1 checks that $\left\{(r^i_{\gamma,\ell}, \mathsf{M}_1[r^i_{\gamma,\ell}], \mathsf{K}_1[r^i_{\gamma,\ell}])\right\}_{i\neq 1}$ are valid and aborts if fails. P_1 computes $z_\gamma \oplus \lambda_\gamma := \bigoplus_{i\in[n]} r^i_{\gamma,\ell}$, and $\left\{\mathsf{L}^i_{\gamma,z_\gamma\oplus\lambda_\gamma} := \mathsf{L}^i_\gamma \oplus \left(\bigoplus_{j\neq i} \mathsf{M}_i[r^j_{\gamma,\ell}]\right)\right\}_{i\neq 1}$

Output Processing:

8. For each $w \in \mathcal{O}, i \neq 1$, P_i sends $(r_w^i, \mathsf{M}_1[r_w^i])$ to P_1, who checks that $(r_w^i, \mathsf{M}_1[r_w^i], \mathsf{K}_1[r_w^i])$ is valid. P_1 computes $z_w := (\lambda_w \oplus z_w) \oplus \left(\bigoplus_{i\in[n]} r_w^i\right)$.

Figure 3: Our main protocol, continued.

4 The Main Scheme

Since we have discussed the main intuition of our protocol, we will proceed to the details directly. The proof of the main protocol can be found in Section A. In Figure 2 and Figure 3 , we present the complete MPC protocol in the $\mathcal{F}_{\mathsf{Pre}}$-hybrid model. The protocol can be divided into five phases:

1. **Circuit Pre-scan.** (Step 1-3) In this phase, each party obtains their own private global MAC keys (Δ_i) from $\mathcal{F}_{\mathsf{Pre}}$, and generate authenticated shares on wire masks for all wires.
2. **Circuit Garbling.** In this phase, each party compute shares of garbled tables for each garbler (Step 4 (a) - 4 (c)). Garblers then compute the distributed garbled circuits based on these shares (Step 4 (d)).
3. **Circuit Input Processing.** For each input wire that corresponds to P_i's input, all other parties reveal their share of the wire mask to P_i. Party[i] then broadcasts the masked input values. All garblers, upon receiving this masked input value, send the corresponding key to the evaluator.
4. **Circuit Circuit Evaluation.** The evaluator evaluate the circuit following the topological order. In detail, the garbled wire labels from each garbler is used to obtain a set of shares of the wire labels for the output of the gate. The wire labels can then be constructed from the shares.

Functionality $\mathcal{F}^n_{\mathsf{aBit}}$

Honest Parties: The box receives $(\mathsf{input}, i, \ell)$ from all parties and picks random bit-string $x \in \{0,1\}^\ell$. For each $j \in [\ell], k \neq i$, the box picks random $\mathsf{K}_k[x_j]$, and computes $\{\mathsf{M}_k[x_j] := \mathsf{K}_k[x_j] \oplus x_j\Delta_k\}_{k\neq i}$, and sends them to parties. That is, for each $j \in [\ell]$, it sends $\{\mathsf{M}_k[x_j]\}_{k\neq i}$ to P_i and sends $\mathsf{K}_k[x_j]$ to P_k for each $k \neq i$.

Malicious Party: Corrupted parties can choose their output from the protocol.

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

Figure 4: Functionality for multi-party authenticated bit.

5. **Circuit Output Processing.** Now the evaluator holds masked output. All garblers reveal their shares of the output wire masks for the circuit to let evaluator unmask the value.

5 Efficiently Realizing $\mathcal{F}_{\mathsf{Pre}}$

In this section, we describe an efficient instantiation of $\mathcal{F}_{\mathsf{Pre}}$, which is a multi-party version of TinyOT protocol. All previous related protocols [LOS14, BLN$^+$15, FKOS15] for multi-party TinyOT rely on cut-and-choose to ensure correctness and another bucketing to ensure privacy, resulting in a communication/computation complexity at least $\Omega(B^2n^2)$ per AND triple, where bucket size $B = \rho/\log|C|$ (See Table 1 for more detail). Furthermore, these protocols output SPDZ-style shares that are not compatible with our main protocol. Our new protocol introduced in this section works with BDOZ-style shares; furthermore the complexity per AND triple is $O(Bn^2)$ with a very small constant. The new protocol features a new distributed AND triple checking protocol that checks the correctness of an AND triple *without* cut-and-choose. The adversary is still able to perform selective failure attacks on a triple with probability of being caught at least one-half. Such leakage can be easily eliminated using bucketing.

In the following, we will build our protocol from bottom up. For some simpler components, we will discuss key ideas in the main body and leave the full protocol and proof to the Appendix. In Section 5.1 and Section 5.2, we discuss the multi-party authenticated bits and authenticated shares that we also introduced in Section 2; in Section 5.3, we discuss an AND triple generation protocol that allows adversary to perform selective failure attack; the finally protocol that eliminates such attack follows the bucketing protocols used in previous works [NNOB12, WRK17], and is detailed in Section B.5.

5.1 Multi-Party Authenticated Bit

The first step of our protocol is to generate multi-party authenticated bit. The functionality $\mathcal{F}^n_{\mathsf{aBit}}$, also discussed in Section 2, is shown in Figure 4. Notice that if we set $n = 2$, then $\mathcal{F}^2_{\mathsf{aBit}}$ is the original two-party authenticated bit functionality [NNOB12]. One naive solution to realize $\mathcal{F}^n_{\mathsf{aBit}}$ is to let P_i run the two-party authenticated bit protocol with every other party using the same bit x. This solution is not secure, since a malicious P_i can potentially use inconsistent values when running $\mathcal{F}^2_{\mathsf{aBit}}$ with other parties. In our protocol, we use this general idea and we also perform additional checks to ensure that P_i uses consistent values. The check is similar to the recent malicious OT extension protocol by Keller et al. [KOS15], where parties perform checks based on random linear combination: a malicious P_i who uses inconsistent values is able to pass ρ checks with probability at most $2^{-\rho}$. Note that these checks also reveal some linear relationship of x's. To eliminate this leakage, a small number of random authenticated bits are computed and checked together. They are later discarded to break the linear relationships. We delay the complete description of the protocol and the proof to Section B.1.

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

Honest Parties: The box receives (input, ℓ) from all parties and picks random bit-strings $x \in \{0,1\}^\ell$ and random authenticated shares $\{\langle x_j \rangle\}_{j \in [\ell]}$, and sends them to parties.
In detail, the box picks random bit strings $\{x^i\}^{i \in [n]}$, each of length ℓ bits. For each $i \in [n], j \in [\ell]$, The box picks random multi-party authenticated bits $[x^i_j]^i$ and sends them to parties. That is, for each $j \in [\ell]$, it sends $(x^i_j, \{\mathsf{M}_k[x^i_j], \mathsf{K}_i[x^k_j]\}_{k \neq i})$ to P_i.

Malicious Party: An corrupted P_i can choose Δ_i and related randomness.

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

Figure 5: Functionality for multi-party authenticated share.

5.2 Multi-Party Authenticated Shares

In this section, we aim to construct a protocol that allows multiple parties to obtain authenticated shares of a secret bit, as shown in Figure 5. One straightforward idea is to call $\mathcal{F}^n_{\mathsf{aBit}}$ n times, where in the i-th execution, they compute $[x^i]^i$ for some random x^i known only to P_i. However, the adversary is still able to perform an attack: a malicious P_i can potentially use different global MAC keys (Δ_i) in different executions of $\mathcal{F}^n_{\mathsf{aBit}}$. The result is that $[x^j]^j$ is authenticated with a global MAC key Δ_i, while some other $[x^k]^k$ is authenticated with a different global MAC key Δ'_i. This attack does not happen in the two-party setting, because each party is authenticated to only one party.

Our key idea is based on the observation that the two-party authenticated bit protocol already ensured that, when P_i and P_j compute multiple authenticated bits, P_i uses the same Δ_i across different authenticated bits. Therefore, in the above insecure attempt, if one authenticated share has consistent global MAC keys, then all authenticated shares have consistent global MAC keys, and vice versa. In our secure construction, we first let all parties compute $\ell + \rho$ number of multi-party authenticated shares as described above, which may not be secure. Then we partially open the last ρ tuples to check the consistency of global MAC keys. A malicious party who uses inconsistent Δ_i's will get caught with probability one-half for each partially opened shares.

In more detail, each player P_i will take the role of a prover once to prove that he uses a consistent Δ_i and the remaining players will take the role of verifier for the given prover. The basic idea is that if the prover used a consistent Δ_i, then these authenticated bits across different parties are XOR homomorphic. Taking a three-party setting as an example. Say P_1 has $\mathsf{K}_1[x]$, $\mathsf{K}_2[y]$ with global keys Δ^x_1 and Δ^y_1 which are potentially different; P_2 has $(x, \mathsf{M}_1[x])$; P_3 has $(y, \mathsf{M}_1[y])$. In our checking protocol, we let P_1 commit to values $\mathsf{K}_1[x] \oplus \mathsf{K}_1[y]$ and $\mathsf{K}_1[x] \oplus \mathsf{K}_1[y] \oplus (x \oplus y)\Delta_1$. For an adversary who uses inconsistent global keys, it needs to choose two values out of the following four values to commit.

$x = 0$	$y = 0$	$\mathsf{K}_1[x] \oplus \mathsf{K}_2[y]$
$x = 0$	$y = 1$	$\mathsf{K}_1[x] \oplus \mathsf{K}_2[y] \oplus \Delta^y_1$
$x = 1$	$y = 0$	$\mathsf{K}_1[x] \oplus \mathsf{K}_2[y] \oplus \Delta^x_1$
$x = 1$	$y = 1$	$\mathsf{K}_1[x] \oplus \mathsf{K}_2[y] \oplus \Delta^x_1 \oplus \Delta^y_1$

Later in the protocol, the adversary is asked to open the MAC for $x \oplus y$. If inconsistent global keys are used, this value can be any of the four values each with one-fourth probability, therefore, the adversary can win with probability at most $2/4 = 1/2$. The details of the protocol and proof are discussed in Section B.2.

5.3 Multi-Party Leaky Authenticated AND Triple

Half authenticated AND triple. Before introducing the protocol for leaky authenticated AND triple, there is yet another tool that we need. As described in Figure 6, the functionality $\mathcal{F}_{\mathsf{HaAND}}$ is introduced

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

1. The box picks random $\langle x \rangle$ and sends it to all parties.
2. Upon receiving $(i, \{y^i_j\}_{j \neq i})$ from all P_i, the box picks random bits $\{v^i\}_{i \in [n]}$ such that $\bigoplus_i v^i := \bigoplus_i \bigoplus_{j \neq i} x^i y^j_i$. The box sends v^i to P_i.

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

Figure 6: The Half Authenticated AND Functionality

to compute cross terms in the AND triple. It takes unauthenticated and potentially inconsistent y's and outputs authenticated share $\langle x \rangle$ as well as unauthenticated shares of cross product terms. Details about the protocol and proof are provided in Section B.3.

Now we are ready to discuss the protocol for leaky authenticated AND triple. It can be divided into following steps:

1. Call $\mathcal{F}_{\mathsf{aShare}}$ to obtain some random $\langle y \rangle$ and $\langle r \rangle$.
2. Call $\mathcal{F}_{\mathsf{HaAND}}$ with y to obtain a random $\langle x \rangle$ and compute shares $\{z^i\}$, such that $(\bigoplus_i x^i) \wedge (\bigoplus_u y^i) = \bigoplus_i z^i$.
3. Reveal $d = z \oplus r$ and computes $\langle z \rangle := \langle r \rangle \oplus d$.
4. Perform additional check to ensure the correctness of the AND relationship.

In the above steps, the adversary is able to cheat by using inconsistent values of y and z between step 1 and 2. However, this only allows the adversary to perform selective failure attack on x^i's. For example, the AND relationship checked is

$$\left(\bigoplus_i x^i\right) \wedge \left(\bigoplus_i y^i\right) = \left(\bigoplus_i z^i\right).$$

The adversary can make guess that the value of $\bigoplus x^i = 0$ and flip y^j for some $j \in \mathcal{M}$. If the guess is correct, than the check will go through the protocol will proceed as normal. However, if the guess is wrong, then the checking will abort and the adversary is caught.

One main challenge is what leakage we should aim for in this functionality. We would like to limit the leakage to be possible only on x^i's, otherwise we would need more bucketing for each possible leakage, as also noted by Nielsen et al. [NNOB12]. On the other hand, the adversary can do more attacks than the one mentioned above: it is also possible to, for example, learn $\bigoplus_{i \in S} x^i$ for some set $S \subset [n]$. We find that the best way to abstract such attack is to allow the adversary to perform a linear check on the value of x^i's. As shown in Figure 7, the adversary is allowed to send a list of coefficients and check if the inner product between the coefficients and x values is zero or not.

Our checking phase differs substantially from existing works. We design an efficient checking protocol, that ensures the correctness of the triple (if no party aborts) which allows malicious parties to learn k bits of some specific information with probability at most 2^{-k}. In the two-party protocol, one party constructs "checking tables" and lets the other party to evaluate/check. In the multi-party protocol here, we instead let all parties distributively construct the "checking tables". Interestingly, distributively constructing these checks is inspired by the main protocol where parties distributively construct garbled tables. As noted before, this protocol is vulnerable to selective failure attacks. The full description of this protocol is presented in Figure 8.

In the following, we will show the correctness and unforgeability of the protocol, which are crucial to the security proof of the protocol.

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

Honest parties: For each $i \in [n]$, the box picks random $\langle x \rangle, \langle y \rangle, \langle z \rangle$ such that $(\bigoplus x^i) \wedge (\bigoplus y^i) = \bigoplus z^i$.

Corrupted parties:

1. Corrupted parties can choose all their randomness.
2. An adversary can send $(Q, \{R_i\}_{i\in[n]})$, which are κ-bit strings, to the box and perform a linear combination test. The box will check
$$Q \oplus \bigoplus_i x^i R_i = 0$$
If the check is incorrect, the box outputs fail and terminates, otherwise the box proceeds as normal.
3. An adversary can also send $(q, \{r_i\}_{i\in[n]})$, which are all bits, to the box and perform a linear combination test. The box will check
$$q \oplus \bigoplus_i x^i r_i = 0$$
If the check is incorrect, the box outputs fail and terminates, otherwise the box proceeds as normal.

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

Figure 7: Functionality $\mathcal{F}_{\mathsf{LaAND}}$ for leaky AND triple generation.

5.3.1 Correctness of the protocol

We want to show that the protocol will compute a correct triple and will not abort if all parties are honest. Notice that the value we are checking can be written as:

$$\begin{aligned}
&\bigoplus_i H_i \\
&= \bigoplus_i \left(x^i \Phi_i \oplus \left(\bigoplus_{k\neq i} \mathsf{K}_i[x^k]_{\Phi_i} \oplus \mathsf{M}_k[x^i]_{\Phi_k} \right) \oplus z^i \Delta_i \oplus \left(\bigoplus_{k\neq i} \mathsf{K}_i[z^k] \oplus \mathsf{M}_k[z^i] \right) \right) \\
&= \bigoplus_i \left(x^i \Phi_i \oplus \left(\bigoplus_{k\neq i} \mathsf{K}_i[x^k]_{\Phi_i} \oplus \mathsf{M}_k[x^i]_{\Phi_k} \right) \right) \oplus \bigoplus_i \left(z^i \Delta_i \oplus \left(\bigoplus_{k\neq i} \mathsf{K}_i[z^k] \oplus \mathsf{M}_k[z^i] \right) \right) \\
&= \bigoplus_i \left(x^i \Phi_i \oplus \left(\bigoplus_{k\neq i} \mathsf{K}_i[x^k]_{\Phi_i} \oplus \mathsf{M}_i[x^k]_{\Phi_k} \right) \right) \oplus \bigoplus_i \left(z^i \Delta_i \oplus \left(\bigoplus_{k\neq i} \mathsf{K}_i[z^k] \oplus \mathsf{M}_i[z^k] \right) \right) \\
&= \bigoplus_i \left(x^i \Phi_i \oplus \left(\bigoplus_{k\neq i} x^k \Phi_i \right) \right) \oplus \bigoplus_i \left(z^i \Delta_i \oplus \left(\bigoplus_{k\neq i} z^k \Delta_i \right) \right) \\
&= \left(\bigoplus_i x^i \right) \cdot \left(\bigoplus_i \Phi_i \right) \oplus \left(\bigoplus_i z^i \right) \cdot \left(\bigoplus_i \Delta_i \right)
\end{aligned}$$

Notice further that

$$\bigoplus_i \Phi_i = \bigoplus_i \left(y^i \Delta_i \oplus \left(\bigoplus_{k\neq i} \mathsf{K}_i[y^k] \oplus \mathsf{M}_k[y^i] \right) \right) = \left(\bigoplus_i y^i \right) \cdot \left(\bigoplus_i \Delta_i \right)$$

Therefore we know that

$$\begin{aligned}
\bigoplus_i H_i &= \left(\bigoplus_i x^i \right) \cdot \left(\bigoplus_i \Phi_i \right) \oplus \left(\bigoplus_i z^i \right) \cdot \left(\bigoplus_i \Delta_i \right) \\
&= \left(\bigoplus_i x^i \right) \cdot \left(\bigoplus_i y^i \right) \cdot \left(\bigoplus_i \Delta_i \right) \oplus \left(\bigoplus_i z^i \right) \cdot \left(\bigoplus_i \Delta_i \right) \\
&= \left(\left(\bigoplus_i x^i \right) \cdot \left(\bigoplus_i y^i \right) \oplus \left(\bigoplus_i z^i \right) \right) \cdot \left(\bigoplus_i \Delta_i \right)
\end{aligned}$$

Since $\bigoplus_i \Delta_i$ is non-zero, $\bigoplus_i H_i = 0$ if and only if the logic of this AND is correct.

Protocol Π_{LaAND}

Triple computation.

1. For each $i \in [n]$ each party calls $\mathcal{F}_{\mathsf{aShare}}$ and obtains random authenticated shares $\{\langle y\rangle, \langle r\rangle\}$. All parties also calls $\mathcal{F}_{\mathsf{HaAND}}$ to obtain random authenticated share $\langle x\rangle$.
2. For each $i \in [n]$, P_i sends $(i, \{y^i\}_{j\neq i})$ to $\mathcal{F}_{\mathsf{HaAND}}$ and gets back some v^i.
3. For each $i \in [n]$, P_i computes $z^i := x^i y^i \oplus v^i$ and $e^i := z^i \oplus r^i$. P_i broadcasts e^i to all other parties. All parties computes $[z^i]^i := [r^i]^i \oplus e^i$.

Triple checking.

4. For each $i \in [n]$, P_i computes: $\Phi_i := y^i\Delta_i \oplus \left(\bigoplus_{k\neq i} \mathsf{K}_i[y^k] \oplus \mathsf{M}_k[y^i]\right)$.
5. For every pair of $i, j \in [n]$, such that $i \neq j$, P_i computes $\mathsf{K}_i[x^j]_{\Phi_i} := H(\mathsf{K}_i[x^j])$ and $U_{i,j} := H(\mathsf{K}_i[x^j] \oplus \Delta_i) \oplus \mathsf{K}_i[x^j]_{\Phi_i} \oplus \Phi_i$, and sends $U_{i,j}$ to P_j. P_j computes $\mathsf{M}_i[x^j]_{\Phi_i} := x^j U_{i,j} \oplus H(\mathsf{M}_i[x^j])$.
6. For $i \in [n]$, P_i computes
   $H_i := x^i\Phi_i \oplus \left(\bigoplus_{k\neq i} \mathsf{K}_i[x^k]_{\Phi_i} \oplus \mathsf{M}_k[x^i]_{\Phi_k}\right) \oplus z^i\Delta_i \oplus \left(\bigoplus_{k\neq i} \mathsf{K}_i[z^k] \oplus \mathsf{M}_k[z^i]\right)$.
   All parties simultaneously broadcast H_i by first broadcasting the commitment of H_i and send the decommitment after receiving commitments from all parties.
7. Each party check if $\bigoplus_i H_i = 0$ and abort if not true.

Figure 8: The protocol Π_{LaAND}.

5.3.2 Unforgeability

Now we want to show that any incorrect AND triple cannot pass the check.

Lemma 5.1. *Define x^i, y^i from $\langle x\rangle, \langle y\rangle$ which are outputs from $\mathcal{F}_{\mathsf{aShare}}$ and $\mathcal{F}_{\mathsf{HaAND}}$; define $z^i := r^i \oplus e^i$, where $\langle r\rangle$ is output from $\mathcal{F}_{\mathsf{aShare}}$, e^i is the value broadcast from P_i. If $\left(\bigoplus_i x^i\right) \wedge \left(\bigoplus_i y^i\right) \neq \left(\bigoplus z^i\right)$ then the protocol results in an abort except with negligible probability.*

We use $U^*_{i,j}$ and H^*_i to denote the values that an honest party would have computed, and define $Q_{i,j} = U^*_{i,j} \oplus U_{i,j}$, $Q_i = H^*_i \oplus H_i$. In the following, we will assume that the logic of the AND does not hold while at the same time that the check passes, and we will derive a contradiction from it.

First note that if P_i uses some $Q_{i,j}$, then P_j will obtain $\mathsf{M}_i[x^j]_{\Phi_i}$ with an additive error of $x^j Q_{i,j}$. Note that

$$\bigoplus_i H^*_i = \left(\left(\bigoplus_i x^i\right) \cdot \left(\bigoplus_i y^i\right) \oplus \left(\bigoplus_i z^i\right)\right) \cdot \left(\bigoplus_i \Delta_i\right) = \bigoplus_i \Delta_i$$

Therefore, we know that

$$\begin{aligned}
\bigoplus_i H_i &= \bigoplus_{i\in\mathcal{M}} H_i \oplus \bigoplus_{i\in\mathcal{H}} H_i \\
&= \bigoplus_{i\in\mathcal{M}} (H^*_i \oplus Q_i) \oplus \bigoplus_{i\in\mathcal{H}} \left(H^*_i \oplus \left(\bigoplus_{k\neq i} x^k Q_{k,i}\right)\right) \\
&= \bigoplus_i H^*_i \oplus \bigoplus_{i\in\mathcal{M}} Q_i \oplus \bigoplus_{i\in\mathcal{H}} \left(\bigoplus_{k\neq i} x^k Q_{k,i}\right) \\
&= \bigoplus_i \Delta_i \oplus \bigoplus_{i\in\mathcal{M}} Q_i \oplus \bigoplus_{i\in\mathcal{H}} \left(\bigoplus_{k\neq i} x^k Q_{k,i}\right)
\end{aligned}$$

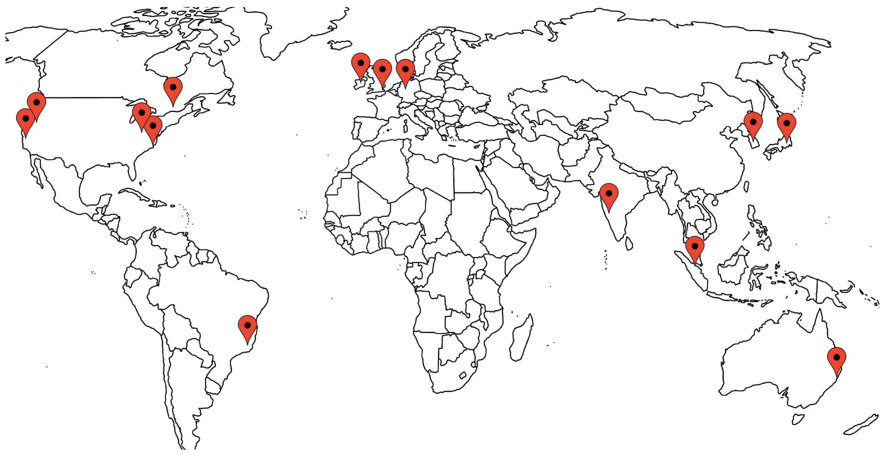


Figure 9: Amazon EC2 regions used in the WAN experiment. Details see Table 7.

Circuit	n_1	n_2	n_3	$\lvert\mathcal{C}\rvert$
AES	128	128	128	6800
SHA-128	256	256	160	37300
SHA-256	256	256	256	90825

Table 4: **Circuits used in our evaluation.**

In order to make $\bigoplus_i H_i$ to be 0, the adversary needs to find paddings such that

$$\bigoplus_{i\in\mathcal{M}} Q_i \oplus \bigoplus_{i\in\mathcal{H}} \left(\bigoplus_{k\neq i} x^k Q_{k,i} \right) = \bigoplus_i \Delta_i$$

The above happens with at most negligible probability.

Theorem 5.1. *Assuming an adversary corrupting up to $n-1$ parties, the protocol in Figure 8, where H is modeled as a random oracle, securely instantiates $\mathcal{F}_{\mathsf{LaAND}}$ functionality in the $(\mathcal{F}_{\mathsf{aShare}}, \mathcal{F}_{\mathsf{HaAND}})$-hybrid model.*

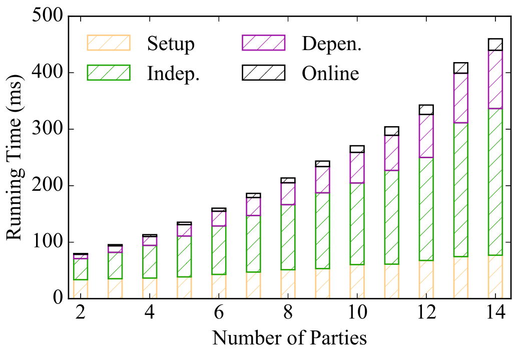
Note that since no party has private input, the simulation proof is straightforward given the lemmas above. The only difficulty is to extract adversary's selective failure attack queries to the ideal functionality. We provide full details of the proof in Section B.4.]

6 Evaluation

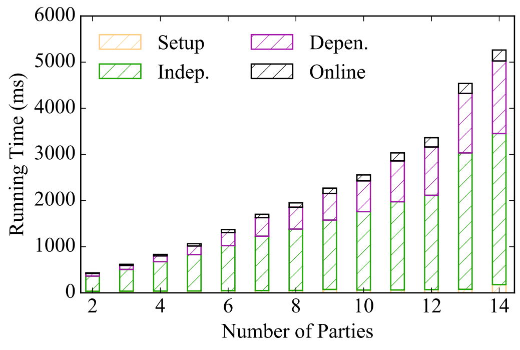
6.1 Evaluation Step

We implemented our protocol in the EMP-toolkit [**?**] framework. We will make our implementation publicly available. To fully explore performance characteristics of our protocol, we evaluate our implementation in three different settings:

- **LAN setting.** Machines are located in the same Amazon EC2 region. Experiments are performed for up to 14 parties.
- **WAN setting.** Each Machine is located in a *different* Amazon EC2 region (locations shown in Figure 9). For a k-party computation experiment, a prefix subset of the machines in Table 7 are selected.

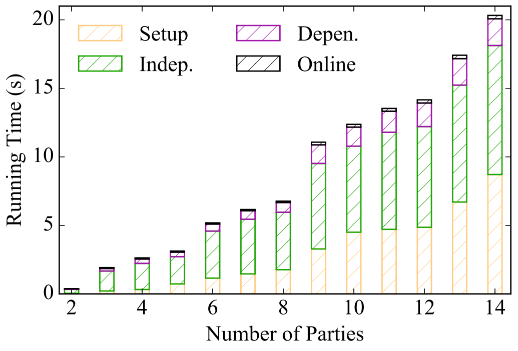


(a) AES evaluation.

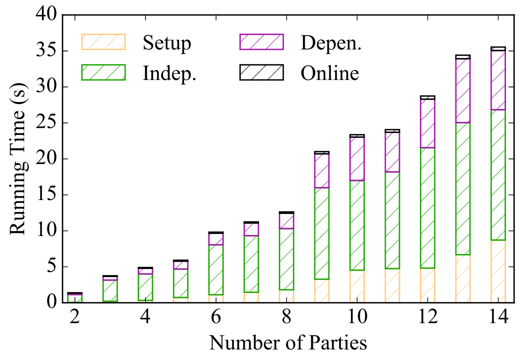


(b) SHA-256 evaluation.

Figure 10: Running time breakdown for basic circuits in the LAN setting.



(a) AES evaluation.



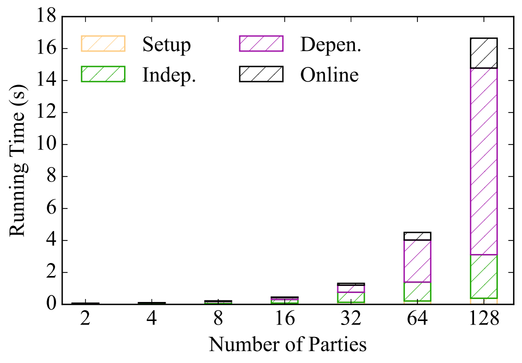
(b) SHA-256 evaluation.

Figure 11: Running time breakdown for basic circuits in the WAN setting. All parties are located in different region worldwide, for example, **2PC**: within US-east; **5PC**: within North America; **8PC**: within North America and Europe; **12PC**: within North America, Europe and Asia; **13PC**: further adds Sydney; **14PC**: all parties in Figure 9.

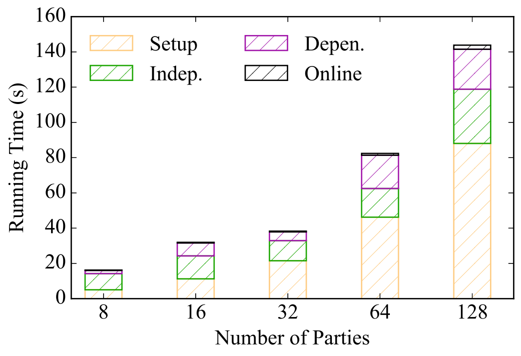
For example, parties in 3PC experiments are located in North Virginia, Ohio, and North California respectively. Experiments are run for up to 14 parties, which is the number of different Amazon EC2 regions available.

- **Crowd setting.** In the LAN case, we evaluate up to 128 parties all located in the same Amazon EC2 region. In the global-scale case, we choose 8 different cities across 5 continents and open up to 16 parties in each city (totally 128 parties).

All machines are of type `c4.8xlarge`, with 36 cores and 60 GB RAM. Network bandwidth within the same region is about 10Gbps. The bandwidth across different regions depends on the location of the machines. All experiments are based on $\rho = 40, \kappa = 128$. We extend justGarble [BHKR13] to support garbling of longer tables in a straightforward manner. In the implementation, we used the "broadcast with abort" protocol by Goldwasser and Lindell [GL05] and achieves the notion of secure computation with abort. We observe small variance when running in the LAN setting and slightly higher variance in the WAN setting. All numbers reported in LAN setting are on average of 10 runs, ones in WAN setting are on average of 20 runs, and ones in the Crowd setting are on average of 5 runs due to the lengthy experiments.



(a) LAN setting. All parties are located in the same region.



(b) Worldwide setting. 8PC is performed with each party located in a different region; 16PC is performed with 2 parties located in each region; others can be interpreted similarly.

Figure 12: Multi-party computation with massive number of parties. Performance based on AES evaluation.

6.2 Performance on Basic Circuits

We evaluate commonly used benchmark circuits on our protocol, including AES, SHA-1, and SHA-256. Information about these circuits can be found in Table 4.

We plot in Figure 10 and Figure 11 the results for AES and SHA-256 with performance breakdown described above. Detailed timings and more results for all three circuits can be found in Table 8 in the Appendix. First, the performance of three-party computation is extremely efficient: it takes 434 ms to evaluate a circuit for SHA-256, with 11.8 ms online time. We also find that in the LAN setting, the slowdown from 2PC to 3PC is roughly 1.5×; the slowdown in the WAN setting is larger. This is caused by the network latency: the first two parties are both located in the U.S. east coast, while the third party is located in the U.S. west coast with much higher latency.

We also find that the cost of the one-time setup is almost independent of number of parties for small number of parties. This is mainly due to the parallelization in the implementation that allows all base-OT to run at the same time.

World-wide MPC experiment. We would like to emphasize that in the case of WAN setting with 14 parties, it is a "world-wide" MPC experiment over 5 continents. To the best of our knowledge, we are the first to conduct MPC over such large range even considering semi-honest MPC protocols.

We also notice a big "jump" in running time from 8 parties to 9 parties in the WAN setting. We believe this is because of the network condition: for experiments up to 8 parties, it is within the US/Europe area; the ninth party is located in asia, where the communication to US/Europe is much slower. More details see Figure 11 and Table 8.

6.3 Evaluation in the Crowd Setting

In this section, we focus on the performance of our protocol with a massive number of parties. Similarly, we summarize the results in Figure 12. We notice that our protocol scales very well with increasing number of parties. Even in a setting with 128 parties located in the same LAN, where up to 127 of them can be corrupted, it takes less than 17 seconds end-to-end running time to compute AES. Note that the performance of a 64-party computation on AES is comparable to the performance of what used to be the state-of-the-art malicious *2-party* computation three years ago [AMPR14], and we belive further optimizations and improvements based on our work will flourish too.

		3	8	16	32
[BLO16]	Online	80	150	400	1500
	Total	228	2000	5900	-
This work	Online	24	95	370	1632
	Total	618	1945	6711	18828

Table 5: Compare with Ben-Efraim et al. [BLO16] Timings are in terms of milliseconds. Their protocol works in the semi-honest setting; ours is maliciously secure. Comparison based on SHA-256, with the same hardware configuration.

When comparing the running time of 128 parties to the one of 8 parties, we find that the increasing of function dependent phase is much higher. This is because our function-independent phase is symmetric and all communication loads are evenly distributed among all parties; while in the function dependent phase, $n-1$ garblers send the garbled circuit to the evaluator, and the bandwidth of the evaluator becomes the bottleneck. Therefore in the case where there are a lot of parties, when we double the number of parties, the running time of the function-dependent phase almost doubles.

We also run the same experiment in the worldwide range. We choose 8 most saperate regions out of 14 and open up to 16 machines in each region (thus totally 128 machines). The performance is also shown in Figure 12: it takes slightly more than a minute for 64 parties to compute AES and about 2.5 minutes for 128 parties located all around the world. We also observe that setup takes much more time; we believe it is due to the high latency.

6.4 Comparison to Related Works

Malicious MPC on AES. Evaluating AES with malicious security against $n-1$ corruption was studied by Damgård et al. [DKL+12]. They reported 240 ms *online time* for 3 parties and 340 ms online time for 10 parties. The offline time for 3 and 10 parties are around 4200 seconds and 15000 seconds respectively. Our protocol takes 95 ms total time to evaluate AES for 3 partis with online time as small as 2 ms; and 268 ms total time with online time 12 ms. The improvement for online phase ranges from $28\times$ to $120\times$; and the improvement for total time ranges from $44000\times$ to $56000\times$. This is a huge improvement even considering hardware differences.

BMR-style protocols. Lindell et al. [LPSY15, LSS16] studied how to use SPDZ and SHE to construct a BMR-style protocol. Since their protocol is not implemented, we compare the communication complexity. After incorporating various optimizations, every AND gate still need $3n+1$ SPDZ multiplication triples. Together with the most recent advance in SPDZ triple generation by Keller et al. [KOS16], generating one SPDZ triple with n parties requires communication about $180(n-1)$ kilobits per party. Therefore the communication cost per AND gate per party is about $540n(n-1)$ kilobits. In our protocol, each AND gate only needs one AND triple from $\mathcal{F}_{\mathsf{Pre}}$, which, using our new protocol in Section 5, requires communication roughly $2.28(n-1)$ kilobits per party. Therefore, the improvement of our protocol compared to the best-optimized BMR protocol based on Lindell et al. is about $237n\times$ with n parties. For a three-party setting, it is an improvement of $711\times$; for the 128-party computation that we perform, the improvement is as high as $30,000\times$!

Ben-Efraim et al. [BLO16] presented a protocol secure in the *semi-honest* model based on BMR. Surprisingly, given the fact that our protocol is maliciously secure, while theirs only has a semi-honest security, our implementation has roughly the same performance as theirs. In Table 5, we compare the running time of our protocol with the running time of theirs based on the same hardware. We notice that for both online time and total time, the performance of the two protocols are roughly the same.

Malicious 3PC protocols with honest majority. Mohassel et al. [MRZ15] proposed an efficient protocol for malicious 3PC with honest majority. Their protocol requires only one garbled circuit to be sent and

	ρ	3	8	16
[HSSV17]	40	14	49	105
	80	55	193	413
This work	40	4.8	16.9	36.4
	80	8.6	30	64.5

Table 6: Compare bandwidth consumption with Hazay et al. [HSSV17]. All numbers are the maximum amount of data one party needs to send in the function-independent phase, measured in terms of megabytes (MB). Numbers for $\rho = 80$ are calculated based on the complexity of both protocols.

therefore has a smaller communication complexity than us. We estimate that our protocol requires about $14\times$ more communication than theirs. However interestingly we also find that the online time of two protocols are roughly the same: their protocol requires 31 ms evaluation time, while ours needs 23.4 ms evaluation time. We belive this is due to the fact that their protocol needs to check that the garbled circuit received from two garblers are the same, while it is not needed in our protocol.

Furukawa et al. [FLNW17] also presented a malicious 3PC protocol with honest majority. Their protocol has a smaller communication overhead compared to the protocol above by Mohassel et al. but requires at least one round of communication per level of the circuit. In addition to a stronger security guarantee that we support dishonest majority, our protocol has a better latency, especially for deep circuit (e.g. SHA-256 has a depth of 4000), while their protocol has a better throughput.

Compare with Hazay et al. [HSSV17]. We also compare with the concurrent work by Hazay et al.. Since their protocol is not implemented, we only compare the bandwidth usage. Due to our improved preprocessing protocol, our function independent cost is much smaller than theirs. In Table 6, we compare the function independent cost of for AES evaluation with different value of ρ. Our protocol uses $3\times$ to $6.5\times$ less communication compared to theirs. We also find that both protocols has similar function-dependent cost and online cost. However, as shown in the previous evaluation, for moderate number of parties, the function-independent cost dominates the overall cost. Therefore the speed up here also translates to the speed up to the whole computation.

6.5 Communication Complexity

In this section, we evaluate the communication complexity of our protocol. All numbers reported here are the maximum amount of data sent from one party. All numbers are obtained by running our implementation, which are slightly higher than calculated values due to implementation details. In Figure 13, we summarize the bandwith use for basic circuits for different number of parties up to 16 parties.

We can see from the figure that the communication required per party grows linearly with the number of parties. In addition, the communication cost of the Setup phase and the Online phase are very small. The total communication cost is dominated by the function-independent phase. Detailed numbers can be found in Table 11 in the Appendix.

Acknowledgments

This material is based upon work supported by NSF awards #1111599 and #1563722; Samuel Ranellucci is also supported by NSF award #1564088. The author would like to thank Roberto Trifiletti, Yan Huang and Ruiyu Zhu for their helpful comments.

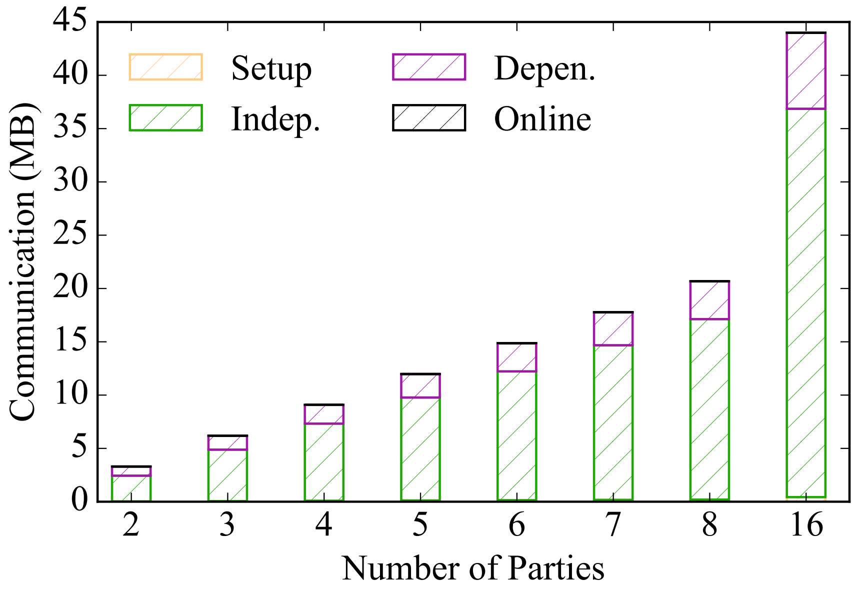


Figure 13: Bandwidth consumption breakdown of our protocol. AES is used. Bandwidth is measured based on the maximum amount of data any party needs to send.

References

[AFL+16] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *ACM CCS 16*, pages 805–817. ACM Press, 2016.

[AMPR14] Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 387–404. Springer, Heidelberg, May 2014.

[BCD+09] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In Roger Dingledine and Philippe Golle, editors, *FC 2009*, volume 5628 of *LNCS*, pages 325–343. Springer, Heidelberg, February 2009.

[BDNP08] Assaf Ben-David, Noam Nisan, and Benny Pinkas. FairplayMP: a system for secure multi-party computation. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 08*, pages 257–266. ACM Press, October 2008.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 169–188. Springer, Heidelberg, May 2011.

[BHKR13] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *2013 IEEE Symposium on Security and Privacy*, pages 478–492. IEEE Computer Society Press, May 2013.

[BLN+15] Sai Sheshank Burra, Enrique Larraia, Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, Emmanuela Orsini, Peter Scholl, and Nigel P. Smart. High performance multi-party computation for binary circuits based on oblivious transfer. Cryptology ePrint Archive, Report 2015/472, 2015. http://eprint.iacr.org/2015/472.

[BLO16] Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Optimizing semi-honest secure multiparty computation for the internet. In *ACM CCS 16*, pages 578–590. ACM Press, 2016.

[BLW08] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In Sushil Jajodia and Javier López, editors, *ESORICS 2008*, volume 5283 of *LNCS*, pages 192–206. Springer, Heidelberg, October 2008.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 503–513. ACM, 1990.

[BSMD10] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics. In Ian Goldberg, editor, *19th USENIX Security Symposium*, Washington, D.C., USA, August 11–13, 2010. USENIX Association.

[CHK$^+$12] Seung Geol Choi, Kyung-Wook Hwang, Jonathan Katz, Tal Malkin, and Dan Rubenstein. Secure multi-party computation of Boolean circuits with applications to privacy in on-line marketplaces. In Orr Dunkelman, editor, *CT-RSA 2012*, volume 7178 of *LNCS*, pages 416–432. Springer, Heidelberg, February / March 2012.

[CKMZ14] Seung Geol Choi, Jonathan Katz, Alex J. Malozemoff, and Vassilis Zikas. Efficient three-party computation from cut-and-choose. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 513–530. Springer, Heidelberg, August 2014.

[DGKN09] Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper Buus Nielsen. Asynchronous multiparty computation: Theory and implementation. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 160–179. Springer, Heidelberg, March 2009.

[DI05] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 378–394. Springer, Heidelberg, August 2005.

[DKL$^+$12] Ivan Damgård, Marcel Keller, Enrique Larraia, Christian Miles, and Nigel P. Smart. Implementing AES via an actively/covertly secure dishonest-majority MPC protocol. In Ivan Visconti and Roberto De Prisco, editors, *SCN 12*, volume 7485 of *LNCS*, pages 241–263. Springer, Heidelberg, September 2012.

[DKL$^+$13] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013*, volume 8134 of *LNCS*, pages 1–18. Springer, Heidelberg, September 2013.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2012.

[FKOS15] Tore Kasper Frederiksen, Marcel Keller, Emmanuela Orsini, and Peter Scholl. A unified approach to MPC with preprocessing using OT. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 711–735. Springer, Heidelberg, November / December 2015.

[FLNW17] Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. High-throughput secure three-party computation for malicious adversaries and an honest majority. In *EUROCRYPT 2017*, LNCS. Springer, Heidelberg, 2017.

[GL05] Shafi Goldwasser and Yehuda Lindell. Secure multi-party computation without agreement. *Journal of Cryptology*, 18(3):247–287, July 2005.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[Gol09] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*, volume 2. Cambridge University Press, 2009.

[HSSV17] Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round mpc combining bmr and oblivious transfer. Cryptology ePrint Archive, Report 2017/214 (Accessed on 03-Mar-2017), 2017. http://eprint.iacr.org/2017/214.

[JMN10] Thomas P. Jakobsen, Marc X. Makkes, and Janus Dam Nielsen. Efficient implementation of the Orlandi protocol. In Jianying Zhou and Moti Yung, editors, *ACNS 10*, volume 6123 of *LNCS*, pages 255–272. Springer, Heidelberg, June 2010.

[KOS15] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 724–741. Springer, Heidelberg, August 2015.

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In *ACM CCS 16*, pages 830–842. ACM Press, 2016.

[KSS13] Marcel Keller, Peter Scholl, and Nigel P. Smart. An architecture for practical actively secure MPC with dishonest majority. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13*, pages 549–560. ACM Press, November 2013.

[LOS14] Enrique Larraia, Emmanuela Orsini, and Nigel P. Smart. Dishonest majority multi-party computation for binary circuits. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 495–512. Springer, Heidelberg, August 2014.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009.

[LPSY15] Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 319–338. Springer, Heidelberg, August 2015.

[LSS16] Yehuda Lindell, Nigel P. Smart, and Eduardo Soria-Vazquez. More efficient constant-round multi-party computation from BMR and SHE. In *TCC 2016-B, Part I*, LNCS, pages 554–581. Springer, Heidelberg, November 2016.

[MRZ15] Payman Mohassel, Mike Rosulek, and Ye Zhang. Fast and secure three-party computation: The garbled circuit approach. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15*, pages 591–602. ACM Press, October 2015.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, Heidelberg, August 2012.

[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.

[WRK17] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. Cryptology ePrint Archive, Report 2017/030, 2017. http://eprint.iacr.org/2017/030.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

A Proof of the Main Protocol

Theorem A.1. *The protocol in Figure 2 and Figure 3, where H is modeled as a random oracle, securely instantiates $\mathcal{F}_{\mathsf{mpc}}$ in the $\mathcal{F}_{\mathsf{Pre}}$-hybrid model with security $\mathsf{negl}(\kappa)$ against an adversary corrupting up to $n-1$ parties.*

Proof. We will consider separately the case where $P_1 \in \mathcal{H}$ and the case where $P_1 \in \mathcal{M}$ and $P_2 \in \mathcal{H}$. The case when $P_1 \in \mathcal{M}$ and $P_i \in \mathcal{H}$ for some $i \geq 3$ is similar to the second case. This covers all cases.

Honest P_1. Let $\mathcal{A}$ be an adversary corrupting $\{P_i\}_{i\in\mathcal{M}}$. We construct a simulator $\mathcal{S}$ that runs $\mathcal{A}$ as a subroutine and plays the role of $\{P_i\}_{i\in\mathcal{M}}$ in the ideal world involving an ideal functionality $\mathcal{F}_{\mathsf{mpc}}$ evaluating f. $\mathcal{S}$ is defined as follows.

1-4 $\mathcal{S}$ acts as honest $\{P_i\}_{i\in\mathcal{H}}$ and plays the functionality of $\mathcal{F}_{\mathsf{Pre}}$, recording all outputs. If any honest party or $\mathcal{F}_{\mathsf{Pre}}$ would abort, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and then aborts.

5 $\mathcal{S}$ interacts with $\mathcal{A}$ acting as an honest $\{P_i\}_{i\in\mathcal{H}}$, using input $\{x^i := 0\}^{i\in\mathcal{H}}$. For each $i \in \mathcal{M}, w \in \mathcal{I}_i$, $\mathcal{S}$ receives $\hat{x}^i_w$ and computes $x^i_w := \hat{x}^i_w \oplus \bigoplus_{i\in[n]} r^i_w$. If any honest party would abort, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and aborts.

6 $\mathcal{S}$ interacts with $\mathcal{A}$ acting as honest $\{P_i\}_{i\in\mathcal{H}}$, using input $x^1 := 0$.

7-8 $\mathcal{S}$ interacts with $\mathcal{A}$ acting as honest $\{P_i\}_{i\in\mathcal{H}}$. If an honest P_1 would abort, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and aborts; otherwise for each $i \in \mathcal{M}$, $\mathcal{S}$ sends (input, x^i) on behalf of P_i to $\mathcal{F}_{\mathsf{mpc}}$.

At any time, $\mathcal{S}$ will answer $\mathcal{A}$'s global key query honestly, since $\mathcal{S}$ knows the global keys of all parties.

Note that since the global keys are randomly selected from $\{0,1\}^\kappa$, $\mathcal{A}$ cannot guess any global key with more than negligible probability. Therefore, in the following, we will assume that it does not happen.

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

Hybrid$_1$. Same as the hybrid-world protocol, where $\mathcal{S}$ plays the role of honest $\{P_i\}_{i\in\mathcal{H}}$, using the actual inputs $\{x^i\}^{i\in\mathcal{H}}$.

Hybrid$_2$. Same as **Hybrid$_1$**, except that in step 5, for each $i \in \mathcal{M}, w \in \mathcal{I}_i$, $\mathcal{S}$ receives $\hat{x}^i_w$ and computes $x^i_w := \hat{x}^i_w \oplus \bigoplus_{i\in[n]} r^i_w$. If any honest party would abort, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs; otherwise for each $i \in \mathcal{M}$, $\mathcal{S}$ sends (input, x^i) on behalf of P_i to $\mathcal{F}_{\mathsf{mpc}}$.

The views produced by the two Hybrids are exactly the same. According to Lemma A.1, P_1 will learn the same output in both Hybrids with all but negligible probability.

Hybrid$_3$. Same as **Hybrid$_2$**, except that, for each $i \in \mathcal{H}$, $\mathcal{S}$ computes $\{r^i_w\}_{w\in\mathcal{I}_i}$ as follows: $\mathcal{S}$ first randomly pick $\{u^i_w\}_{w\in\mathcal{I}_i}$, and then computes $r^i_w := u^i_w \oplus x^i_w$.

The two Hybrids produce exactly the same view.

Hybrid$_4$. Same as **Hybrid$_3$**, except that $\mathcal{S}$ uses $\{x^i = 0\}^{i\in\mathcal{H}}$ as input in step 5 and step 6.

Note that although the distribution of $\{x^i\}^{i\in\mathcal{H}}$ in **Hybrid$_3$** and **Hybrid$_4$** are different, the distribution of $\{x^i_w \oplus r^i_w\}^{i\in\mathcal{H}}$ are exactly the same. The views produced by the two Hybrids are therefore the same, we will show that P_1 aborts with the same probability in both Hybrids.

Observe that the only place where P_1's abort can possibly depends on $\{x^i\}^{i\in\mathcal{H}}$ 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 Hybrids.

As **Hybrid$_4$** is the ideal-world execution, this completes the proof when P_1 is honest.

Malicious P_1 and honest P_2. Let $\mathcal{A}$ be an adversary corrupting $\{P_i\}_{i\in\mathcal{M}}$. We construct a simulator $\mathcal{S}$ that runs $\mathcal{A}$ as a subroutine and plays the role of $\{P_i\}_{i\in\mathcal{M}}$ in the ideal world involving an ideal functionality $\mathcal{F}_{\mathsf{mpc}}$ evaluating f. $\mathcal{S}$ is defined as follows.

1-4 $\mathcal{S}$ acts as honest $\{P_i\}_{i\in\mathcal{H}}$ and plays the functionality of $\mathcal{F}_{\mathsf{Pre}}$, recording all outputs. If any honest party would abort, $\mathcal{S}$ output whatever $\mathcal{A}$ outputs and aborts.

5-6 $\mathcal{S}$ interacts with $\mathcal{A}$ acting as honest $\{P_i\}_{i\in\mathcal{H}}$, using input $\{x^i := 0\}^{i\in\mathcal{H}}$. For each $i \in \mathcal{M}, w \in \mathcal{I}_i$, $\mathcal{S}$ receives $\hat{x}^i_w$ and computes $x^i_w := \hat{x}^i_w \oplus \bigoplus_{i\in[n]} r^i_w$. If any honest party would abort, $\mathcal{S}$ output whatever $\mathcal{A}$ outputs and aborts.

8 For each $i \in \mathcal{M}$, $\mathcal{S}$ sends (input, x^i) on behalf of P_i to $\mathcal{F}_{\mathsf{mpc}}$. If $\mathcal{F}_{\mathsf{mpc}}$ abort, $\mathcal{S}$ aborts, outputting whatever $\mathcal{A}$ outputs. Otherwise, if $\mathcal{S}$ receives z as the output, $\mathcal{S}$ computes $z' := f(y^1, ..., y^n)$, where $\{y^i := 0\}^{i\in\mathcal{H}}$, and $\{y^i := x^i\}^{i\in\mathcal{M}}$. For each $i \in \mathcal{H}, w \in \mathcal{O}$, if $z'_w = z_w$, $\mathcal{S}$ sends $(r^i_w, \mathsf{M}_1[r^i_w])$ on behalf of P_i to $\mathcal{A}$; otherwise, $\mathcal{S}$ sends $(r^i_w \oplus 1, \mathsf{M}_1[r^i_w] \oplus \Delta_1)$.

At any time, $\mathcal{S}$ will answer $\mathcal{A}$'s global key query honestly, since $\mathcal{S}$ knows the global keys of all parties.

Note that since the global keys are randomly selected from $\{0,1\}^\kappa$, $\mathcal{A}$ cannot guess any global key with more than negligible probability. Therefore, in the following, we will assume it does not happen.

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

Hybrid$_1$. Same as the hybrid-world protocol, where $\mathcal{S}$ plays the role of honest $\{P_i\}_{i\in\mathcal{H}}$ using the actual inputs $\{x^i\}^{i\in\mathcal{H}}$.

Hybrid$_2$. Same as **Hybrid$_1$**, except that in step 5 and step 6, for each $i \in \mathcal{M}, w \in \mathcal{I}_i$, $\mathcal{S}$ receives $\hat{x}^i_w$ and computes $x^i_w := \hat{x}^i_w \oplus \bigoplus_{i\in[n]} r^i_w$. If any honest party would abort, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs; otherwise for each $i \in \mathcal{M}$, $\mathcal{S}$ sends (input, x^i) on behalf of P_i to $\mathcal{F}_{\mathsf{mpc}}$.

P_1 does not have output; furthermore the view of $\mathcal{A}$ does not change between the two Hybrids.

Hybrid$_3$. Same as **Hybrid$_2$**, except that in step 5 and step 6, $\mathcal{S}$ uses $\{x^i := 0\}^{i\in\mathcal{H}}$ as input and in step 8, $\mathcal{S}$ computes z' as defined. For each $w \in \mathcal{O}$, if $z'_w = z_w$, $\mathcal{S}$ sends $(r^i_w, \mathsf{M}_1[r^i_w])$; otherwise, $\mathcal{S}$ sends $(r^i_w \oplus 1, \mathsf{M}_1[r^i_w] \oplus \Delta_1)$.

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

Note that since $\mathcal{S}$ uses different values for x between the two Hybrids, we also need to show that the distribution of garbled rows opened by P_1 are indistinguishable for the two Hybrids. According to Lemma A.2, P_1 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 P_1, masked values and garbled keys are indistinguishable between two Hybrids.

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

Lemma A.1. *Consider an $\mathcal{A}$ corrupting parties $\{P_i\}_{i\in\mathcal{M}}$ such that $P_1 \in \mathcal{H}$, and denote $x^i_w := \hat{x}^i_w \oplus \bigoplus_{i=1}^n r^i_w$, where $\hat{x}_w$ is the value $\mathcal{A}$ sent, r^i_w are the values from $\mathcal{F}_{\mathsf{Pre}}$. With probability all but negligible, P_1 either aborts or learns $z = f(x^1, ..., x^n)$.*

Proof. Define z^*_w as the correct wire values computed using x defined above and y, z_w as the actually wire values P_1 holds in the evaluation.

We will first show that P_1 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 P_1 has $\{z^*_w \oplus \lambda_w = z_w \oplus \lambda_w\}_{w\in\{\alpha,\beta\}}$, then P_1 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, P_1 will open the garbled row defined by $i := 2(z_\alpha \oplus \lambda_\alpha) + (z_\beta \oplus \lambda_\beta)$. If P_1 learns $z_\gamma \oplus \lambda_\gamma \neq z^*_\gamma \oplus \lambda_\gamma$, then it means that P_1 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 P_1 learns correct masked output. P_1 can therefore learn correct output $f(x, y)$ unless $\mathcal{A}$ is able to flip $\{r_w\}_{w\in\mathcal{O}}$, which, again, only happens with negligible probability. □

Lemma A.2. *Consider an $\mathcal{A}$ corrupting $\{P_i\}_{i\in\mathcal{M}}$ and that $P_1 \in \mathcal{M}$, with negligible probability, P_1 learns both garbled labels for some wire generated by an honest party.*

Proof. The proof is very similar to the proof of semi-honest garbled circuit protocol by Lindell and Pinkas [LP09]. Let's use z^*_w's to denote the correct value on all input wire and internal wires if x and y defined above are used to evaluate the circuit, and use z_w to denote the actual wire values when P_1 is malicious.

We will show that $z^*_w \oplus \lambda_w = z_w \oplus \lambda_w$, and $\mathsf{L}_{w,z^*_w\oplus\lambda_w} = \mathsf{L}_{w,z_w\oplus\lambda_w}$, and that P_1 does not learn $\mathsf{L}_{w,z_w\oplus\lambda_w\oplus 1}$ for all $w \in \mathcal{O}$.

Base step: Honest P_i only sends one garbled labels to P_1, and Δ_i is hidden from $\mathcal{A}$, therefore the base step is true.

Induction step: It is obvious that P_1 cannot learn the other label for an XOR gate and we will focus on AND gates.

Note that P_1 only learns one garbled keys for input wire α and β. However, each row is encrypted using different combinations of $L_{\alpha,b}$ and $\mathsf{L}_{\beta,b}$. In order for P_1 to open two rows in the garbled table, P_1 needs to learn both garbled keys for some input wire, which contradict with assumptions in the induction step. □

B Instantiation of the Preprocessing Functionality

In this section, we describe an efficient instantiation of $\mathcal{F}_{\mathsf{Pre}}$. All previous protocols [LOS14, BLN+15, FKOS15] for multi-party TinyOT relies on cut-and-choose with bucketing to ensure correctness and at least an additional round of bucketing to ensure privacy, resulting in complexity at least $O(B^2n^2)$ per AND triple, where B is the bucket size. In order to achieve better performance, we instead propose a new distributed checking protocol that allows parties to distributively check the correctness of *each* triple, without cut-and-choose. The adversary is able to perform selective failure attacks on a triple where the probability of being caught is at least one-half. We then used bucketing to eliminate such leakage. Overall our protocol has complexity $O(Bn^2)$.

B.1 Multi-Party Authenticated Bit

The first step of our protocol is to design a multi-party variant of authenticated bit [NNOB12]. One naive solution for P_i to obtain an authenticated bit is to let P_i run a two-party authenticated bit protocol ($\mathcal{F}^2_{\mathsf{aBit}}$) with every other party using the same input x. This solution does not work, since a malicious P_i can

Protocol Π^n_{aBit}

1. Set $\ell' := \ell + 2\rho$. P_i picks random bit-string $x \in \{0,1\}^{\ell'}$.
2. For each $k \neq i$, P_i and P_k runs $\mathcal{F}^2_{\mathsf{aBit}}$, where P_i sends $\{x_j\}_{j\in[\ell']}$ to $\mathcal{F}^2_{\mathsf{aBit}}$. From the functionality, P_i gets $\{\mathsf{M}_k[x_j]\}_{j\in[\ell']}$, P_k gets $\{\mathsf{K}_k[x_j]\}_{j\in[\ell']}$.
3. For $j \in [2\rho]$, all parties perform the following:
   (a) All parties sample a random ℓ'-bit strings r.
   (b) P_i computes $\mathcal{X}_j = \bigoplus_{m=1}^{\ell'} r_m x_m$, and broadcast $\mathcal{X}_j$, and computes $\left\{\mathsf{M}_k[\mathcal{X}_j] = \bigoplus_{m=1}^{\ell'} r_m \mathsf{M}_k[x_m]\right\}_{k\neq i}$.
   (c) P_k computes $\mathsf{K}_k[\mathcal{X}_j] = \bigoplus_{m=1}^{\ell'} r_m \mathsf{K}_k[x_m]$.
   (d) P_i sends $\mathsf{M}_k[\mathcal{X}_j]$ to P_k who check the validity.
4. All parties return the first ℓ objects.

Figure 14: The protocol Π^n_{aBit} instantiating $\mathcal{F}^n_{\mathsf{aBit}}$.

potentially use inconsistent values when running $\mathcal{F}^2_{\mathsf{aBit}}$ with other parties. In our protocol shown in Figure 14, we use this general idea and in addition, we also perform checks to ensure that the values are consistent. The check is similar to the recent malicious OT extension protocol by Keller et al. [KOS15], where parties perform some random linear checks, which reveals some linear relationship among the inputs. To eliminate such leakage, a small number of additional random authenticated bits are computed and checked together. They are later discarded to break the linear dependency.

Theorem B.1. *The protocol in Figure 14 securely instantiates the $\mathcal{F}^n_{\mathsf{aBit}}$ functionality with statistical security $2^{-\rho}$ in the $\mathcal{F}^2_{\mathsf{aBit}}$-hybrid model.*

Proof. **Case 1: $P_i \in \mathcal{H}$.** Note that in this case, the only way malicious parties can break the protocol is by learning some information about $\{x_i\}_{i\in[\ell]}$ in the checking step. However, we will show that, because we "throw out" the last 2ρ authenticated bits, the adversary can learn nothing about x's.

We use s_j to denote the last 2ρ bits of r in the j-th check. According to Lemma B.1 and the parameters we chose, the probability that any subset of $\{s_j\}_{j\in[2\rho]}$ is linearly independent is $1-2^{-\rho}$. Now we will show that if linear independence holds then the adversary cannot learn anything.

For the j-checking, $\mathcal{X} = \left(\bigoplus_{m=1}^{\ell} r_m x_m\right) \oplus \left(\bigoplus_{m=1}^{2\rho} s_m x_{\ell+m}\right)$. Note that $\bigoplus_{m=1}^{2\rho} s_m x_{\ell+m}$ from each checking are independent random bits, where $\{x_m\}_{m=\ell}^{\ell'}$ is random. This is true because the s_i's are linearly independent. Therefore, $\bigoplus_{m=1}^{2\rho} s_m x_{\ell+m}$ acts as one-time pad to $\bigoplus_{m=1}^{\ell} r_m x_m$. Given the above, the simulation is straightforward. Note that for all global key queries, $\mathcal{S}$ can answer them honestly, since $\mathcal{S}$ knows the global key for both parties.

Case 2: $P_i \in \mathcal{M}$. The simulation is straightforward if we could show that for any $\mathcal{A}$ who uses inconsistent x's can pass all 2ρ checks with at most negligible probability. This is what we will proceed to show.

Suppose that $\mathcal{A}$ sends x^1 to $\mathcal{F}^2_{\mathsf{aBit}}$ when interacting with one honest party, and uses a different x^2 with another honest party, where $x^1 \neq x^2$. We also assume that $\mathcal{A}$ passes all checks. Note that for the j-th checking, if $\mathcal{A}$ is not able to forge a MAC, then the probability that the checking passes is the probability that $\mathcal{X}_j = \bigoplus_m r_m x_m^1$ and that $\mathcal{X}_j = \bigoplus_m r_m x_m^2$.

$$\begin{aligned}
&\Pr\left\{\bigoplus_m r_m x_m^1 = \bigoplus_m r_m x_m^2\right\}\\
&= \Pr\left\{\bigoplus_m r_m(x_m^1 \oplus x_m^2) = 0\right\}\\
&= \Pr\left\{\bigoplus_{m\in I} r_m = 0 : I \text{ is the set of indices where } x_m^1 \neq x_m^2\right\}\\
&= 1/2
\end{aligned}$$

Each checking is independent as long as r is selected independently. Therefore, $\mathcal{A}$ can pass all checks with probability at most $2^{-2\rho}$. □

Lemma B.1. *Let $r_1, ..., r_l$ be random bit vectors of length k. With probability at most 2^{l-k}, there exists some subset $I \subset [l]$, such that*

$$\bigoplus_{i\in I} r_i = 0$$

Proof. Note that given a fixed interval $I \subset [l]$, the probability that $\bigoplus_{i\in I} r_i = 0$ is 2^{-k}. According to the union bound, the probability that any subset $I \subset [l]$ has $\bigoplus_{i\in I} r_i = 0$ is $2^{-k} \times 2^l = 2^{l-k}$. □

B.2 Multi-Party Authenticated Share

Note that the above functionality prevents a malicious party from using different x's when computing authenticated bits MACed by different parties. However, it is still possible that a malicious party uses inconsistent Δ's when authenticating different parties' shares. In order to prevent this, we perform additional consistency checks.

From a high level idea, we have already ensured that when P_i and P_j computes authenticated bits, P_i will use consistent Δ_i. Therefore, we will perform the batch checking: after all parties computes $\ell's$ number of multi-party authenticated bits for each of their value, we let them open the last ρ tuples and use these values to validate the consistency of Δ's.

Each player will take the role of a prover once to prove that he uses a consistent Δ and the remaining players will take the role of the verifier for the given prover. The basic idea is that if the prover used a consistent Δ, then the prover can compute a key for which the verifiers hold a shared MAC on the parity of bits. Otherwise, the prover can only try to guess the values received and the following check will fail The idea is that the prover will commit to two values: the first value that he will reveal is the xor of the keys used for that value, the second value is the xor of the keys xored with his Δ. This will hide Δ. Later on, based on the parity of bits, the prover will open one of the commitments and the verifiers will verify that it matches the xor of MACs that they received. To ensure that malicious verifiers cannot help the prover cheat, we will force each verifier to broadcast a commitment to the MACs and values he received before any verifier reveals the bit he authenticated. See Figure 15 for more details.

Theorem B.2. *The protocol in Figure 15 securely instantiates the $\mathcal{F}_{\mathsf{aShare}}$ functionality with statistical security $2^{-\rho}$ in the $\mathcal{F}^n_{\mathsf{aBit}}$-hybrid model.*

Proof. Without loss of generality, we will prove for the case when $P_1 \in \mathcal{H}$, that is P_1 is honest.

The simulator plays the role of $\mathcal{F}^n_{\mathsf{aBit}}$ honestly, recording all values it sends to $\mathcal{A}$ and values $\mathcal{A}$ sent to $\mathcal{F}^n_{\mathsf{aBit}}$. $\mathcal{S}$ acts as honest parties and check for each $i \in \mathcal{M}$, if P_i sent consistent Δ_i in all instructions to $\mathcal{F}^n_{\mathsf{aBit}}$. If not, $\mathcal{S}$ aborts outputting whatever $\mathcal{A}$ outputs.

Note that this simulator has a $2^{-\rho}$ statistical difference to the real world execution given Lemma B.2 □

Protocol Π_{aShare}

1. Set $\ell' := \ell + \rho$. For each $i \in [n]$, P_i picks random bit-string $x^i \in \{0,1\}^{\ell'}$.
2. For each $i \in [n]$, all parties compute multi-party authenticated bits by sending (i, ℓ') to $\mathcal{F}^n_{\mathsf{aBit}}$, which sends $\{[x^i_j]^i\}_{j\in[\ell']}$ to parties.
3. For $r \in [\rho]$, all parties perform the following:
   (a) For each $i \in [n]$, P_i parses $\{[x^k_{\ell+r}]^k\}_{k\in[n]}$ as $(x^i_{\ell+r}, \{\mathsf{M}_k[x^i_{\ell+r}], \mathsf{K}_i[x^k_{\ell+r}]\}_{k\neq i})$. Each P_i computes commitments $(\mathsf{c}^0_i, \mathsf{d}^0_i) \leftarrow \mathsf{Com}(\bigoplus_{k\neq i} \mathsf{K}_i[x^k_{\ell+r}])$, $(\mathsf{c}^1_i, \mathsf{d}^1_i) \leftarrow \mathsf{Com}(\bigoplus_{k\neq i} \mathsf{K}_i[x^k_{\ell+r}] \oplus \Delta_i)$, and $(\mathsf{c}^M_i, \mathsf{d}^M_i) \leftarrow \mathsf{Com}(x^i_{\ell+r}, \{\mathsf{M}_k[x^i_{\ell+r}]\}_{k\neq i})$, and broadcast $(\mathsf{c}^m_i, \mathsf{c}^0_i, \mathsf{c}^1_i)$.
   (b) For each $i \in [n]$, after receiving all commitments, P_i broadcasts d^M_i.
   (c) For each $i \in [n]$, P_i computes $b^i := \bigoplus_{k\neq i} x^k_{\ell+r}$, and broadcast $\mathsf{d}^{b^i}_i$.
   (d) For each $i \in [n]$, P_i performs the following to check the consistency of Δ's: For each $j \neq i$, P_i computes $K^j \leftarrow \mathsf{Open}(\mathsf{c}_{b^j}, \mathsf{d}_{b^j})$ and check if it equals to $\bigoplus_{k\neq j} \mathsf{M}_j[x^k_{\ell+r}]$. If any check fails, the party aborts.
4. All parties return the first ℓ objects.

Figure 15: The protocol Π_{aShare} instantiating $\mathcal{F}_{\mathsf{aShare}}$.

Lemma B.2. *When a malicious P_i computes MACs with P_j, denote Δ^j_i as the value P_i sent to $\mathcal{F}^n_{\mathsf{aBit}}$. If for some $\Delta^{j1}_i \neq \Delta^{j2}_i$, the protocol abort with probability at least $1 - 2^{-\rho}$.*

Proof. In the following, we will prove that malicious party passes each single test with probability 1/2, independently. Since malicious parties are ensured to use the same Δ for each party, it can either cheat for all bits or be honest for all bits. Therefore cheating malicious parties cannot pass all checks with more than $2^{-\rho}$ probability.

We will prove by contradiction. Suppose at least one party uses inconsistent value and the check passes. We use K^i to denote the value P_i opened in step 3 (d), and use $\{M^i_k\}_{k\neq i}$ to denote the value committed in step 3 (c). First compute $Q^i := K^i \oplus \bigoplus_{k\neq i} \mathsf{K}_i[x^k_{\ell+r}]$ and and $Q^i_k = M^i_k \oplus \mathsf{M}_k[x^i_{\ell+r}]$. Since the check for P_i passes, we know that the following is zero.

$$\begin{aligned}
K^i \oplus \bigoplus_{k\neq i} M^k_i &= \left(\bigoplus_{k\neq i} \mathsf{K}_i[x^k_{\ell+r}]\right) \oplus Q^i \oplus \left(\bigoplus_{k\neq i} \mathsf{M}_i[x^k_{\ell+r}] \oplus Q^k_i\right) \\
&= \left(\bigoplus_{k\neq i} \mathsf{K}_i[x^k_{\ell+r}]\right) \oplus Q^i \\
&\quad \oplus \left(\bigoplus_{k\neq i} \mathsf{K}_i[x^k_{\ell+r}] \oplus x^k_{\ell+r}\Delta^k_i\right) \oplus \bigoplus_{k\neq i} Q^k_i \\
&= \left(Q^i \oplus \bigoplus_{k\neq i} Q^k_i\right) \oplus \left(\bigoplus_{k\neq i} x^k_{\ell+r}\Delta^k_i\right)
\end{aligned}$$

If P_i uses $\Delta^{k1}_{\ell+r} \neq \Delta^{k2}_{\ell+r}$ for some $k1 \neq k2$, $k1, k2 \in \mathcal{H}$, then the value of $\left(Q^i \oplus \bigoplus_{k\neq i} Q^k_i\right)$ that makes the equation as 0 is different depending on the value of $x^{k1}_{\ell+r}$ and $x^{k2}_{\ell+r}$. This means that P_i needs to guess at least one of them to pass the check. □

Protocol Π_{HaAND}

1. All parties call $\mathcal{F}_{\mathsf{aShare}}$ to obtain $\langle x \rangle$.
2. For each $i, j \in [n]$, such that $i \neq j$,
   (a) P_i picks a random bit s^j, and computes $H_0 := \mathsf{Lsb}(H(\mathsf{K}_i[x^j])) \oplus s^j$, $H_1 = \mathsf{Lsb}(H(\mathsf{K}_i[x^j] \oplus \Delta_i)) \oplus s^j \oplus y^i_j$.
   (b) P_i sends (H_0, H_1) to P_j, who computes $t^i := H_{x^j} \oplus \mathsf{Lsb}(H(\mathsf{M}_i[x^j]))$.
3. For each $i \in [n]$, P_i obtains $v^i := \bigoplus_{k \neq i}(t^k \oplus s^k)$.

Figure 16: Protocol Π_{HaAND} instantiating $\mathcal{F}_{\mathsf{HaAND}}$.

B.3 Multi-Party Half Authenticated AND triple

Lemma B.3. *Assuming H as a random oracle, the protocol in Figure 16 securely implements the functionality in Figure 6 in the $\mathcal{F}_{\mathsf{aShare}}$-hybrid model.*

Proof. Note that for each $i \in [n]$, P_i has values $\{s^j, t^j\}_{j \neq i}$. We denote the value s^j, t^j held by P_i as s^j_i, t^j_i.
Correctness. First we will show the correctness of the protocol. We further first show that for any $i \neq j$, $s^j_i \oplus t^i_j = x^j y^i_j$. We will discuss in two cases:

- $x^j = 0$. In this case, P_j obtains $t^i_j = s^j$.
- $x^j = 1$. In this case, P_j obtains $t^i_j = s^j \oplus y^i_j$.

In any case, the above equation holds. Now the correctness of the protocol can be seen given the following equation.

$$\begin{aligned}
\bigoplus_i \bigoplus_{j \neq i} x^i y^j_i &= \bigoplus_i \bigoplus_{j \neq i} (s^j_i \oplus t^i_j) \\
&= \bigoplus_i \bigoplus_{j \neq i} s^j_i \oplus \bigoplus_i \bigoplus_{j \neq i} t^i_j \\
&= \bigoplus_i \bigoplus_{j \neq i} s^i_j \oplus \bigoplus_i \bigoplus_{j \neq i} t^i_j \\
&= \bigoplus_i \left(\bigoplus_{j \neq i} s^i_j \oplus t^i_j \right) \\
&= \bigoplus_i v^i
\end{aligned}$$

Simulation Proof. We will prove the security assuming P_1 is honest, that is, $P_1 \in \mathcal{H}$. The simulation is as follows:

1. $\mathcal{S}$ plays the role of $\mathcal{F}_{\mathsf{aShare}}$ storing all values used.
2. For each pair $i \neq j$, such that $i \in \mathcal{M}$, $\mathcal{S}$ obtains (H_0, H_1) sent by malicious P_i. $\mathcal{S}$ computes $s^j := H_0 \oplus \mathsf{Lsb}(H(\mathsf{K}_i[x^j]))$ and $y^i_j := H_1 \oplus \mathsf{Lsb}(H(\mathsf{K}_i[x^j] \oplus \Delta_i)) \oplus s^j$. For each $i \in \mathcal{M}$, $\mathcal{S}$ sends $(i, \{y^i_j\}_{j \neq i})$ to $\mathcal{F}_{\mathsf{HaAND}}$, which sends back $\{v^i\}_{i \in \mathcal{M}}$.
3. For each $i \in \mathcal{M}$, $\mathcal{S}$ picks random $\{t'^k\}_{k \in \mathcal{H}}$, such that $\bigoplus_{k \neq i} s^i \oplus \bigoplus_{k \neq i, k \in \mathcal{M}} t^k_i \oplus \bigoplus_{k \neq i, k \in \mathcal{H}} t'^i = v^i$. For each $j \in \mathcal{H}$, $\mathcal{S}$ computes $H_{x^i} = \mathsf{Lsb}(H(\mathsf{K}[x_i] \oplus x_i \Delta_j)) \oplus t'^j$, and picks a random $H_{1 \oplus x^i}$. $\mathcal{S}$ sends (H_0, H_1) to P_i on behalf of an honest P_j.

First of all, the first two steps are perfect simulation. For the last step, it is also a perfect simulation: first the one that is not opened is random since H is a random oracle. The other value is also random, depending on the value of s^j. However, in order to make the joint distribution of the value $\mathcal{A}$ learns here and the output of an honest P_2 indistinguishable between ideal and real world protocol, t^k are tweaked such that $\mathcal{A}$ will learn the same value in both Hybrids. ☐

B.4 Multi-Party Leaky Authenticated AND Triple

We have described the protocol and the key ideas of the proof in the main body. Here we will directly proceed to the proof.

Theorem B.3. *The protocol in Figure 8, where H is modeled as a random oracle, securely instantiates $\mathcal{F}_{\mathsf{LaAND}}$ functionality in the $(\mathcal{F}_{\mathsf{aShare}}, \mathcal{F}_{\mathsf{HaAND}})$-hybrid model.*

Proof. We constructor a simulator in the following. For all global key queries, $\mathcal{S}$ redirect them to $\mathcal{F}_{\mathsf{aShare}}$ and redirect the answer to $\mathcal{A}$.

1. $\mathcal{S}$ plays the role of $\mathcal{F}_{\mathsf{aShare}}$ storing all information sent to parties.

2-3 $\mathcal{S}$ obtains $(i, \{y^i_j\}_{j\neq i})$ for each $P_i \in \mathcal{M}$. $\mathcal{S}$ also obtains $\{e^i\}_{i\in\mathcal{M}}$ $\mathcal{A}$ broadcasts. $\mathcal{S}$ first computes e^{*i}, which are what an honest P_i would have broadcast and compute $q_i := e^i \oplus e^{*i}$. $\mathcal{S}$ further computes $r_{i,j} := y^i_j \oplus y^i$, where y^i is the value $\mathcal{S}$ used when playing the role of $\mathcal{F}_{\mathsf{aShare}}$. $\mathcal{S}$ computes $r_i := \bigoplus_{j\in\mathcal{M},j\neq i} r_{j,i}$ $q := \bigoplus_{i\in\mathcal{M}} q_i$, and sends $(q, \{r_i\}_n)$ to $\mathcal{F}_{\mathsf{LaAND}}$. If $\mathcal{F}_{\mathsf{LaAND}}$ terminates, $\mathcal{S}$ follows the protocol as honest parties and abort in step 7.

4-5. For each $i \in \mathcal{M}$, $\mathcal{S}$ receives $\{U_{i,j}\}_{j\in\mathcal{H}}$ from P_i. $\mathcal{S}$ picks random $\{U_{j,i}\}_{j\in\mathcal{H}}$ and sends them to P_i playing the role of P_j for each $j \in \mathcal{H}$.

6-7 If $\mathcal{F}_{\mathsf{LaAND}}$ terminate in step 2, then $\mathcal{S}$ follows the protocol as honest parties and abort in step 7. If the equation hold, $\mathcal{S}$ will extract another selective failure attack query.

Similar to the unforgeability proof, we use $U^*_{i,j}$ and H^*_i to denote the values that an honest party would have compute, and define $Q_{i,j} = U^*_{i,j} \oplus U_{i,j}$, $Q_i = H^*_i \oplus H_i$. This means that is a malicious P_i uses some $Q_{i,j}$, then P_j will obtain some $\mathsf{M}_i[x^j]_{\Phi_i}$ with an additive error of $x^j Q_{i,j}$. $\mathcal{S}$ defines $R_k = \bigoplus_{i\neq k, i\in\mathcal{H}} Q_{k,i}$. $\mathcal{S}$ sends $(\bigoplus_{i\in\mathcal{M}} Q_i, \{R_i\}_{i\in[n]})$ to $\mathcal{F}_{\mathsf{LaAND}}$. If $\mathcal{F}_{\mathsf{LaAND}}$ terminates, $\mathcal{S}$ aborts outputting whatever $\mathcal{A}$ outputs; otherwise, $\mathcal{S}$ obtains $\{H_i\}_{i\in\mathcal{M}}$ and picks random $\{H_i\}_{i\in\mathcal{H}}$ such that $\bigoplus_i H_i = 0$.

Note that the first five steps are perfectly indistinguishable given that H is a random oracle, except that $\mathcal{A}$ can perform a selective failure attack. We will show that the probability of abort due to this attack is the same between real-world protocol and ideal-world protocol. The probability that the value $\mathcal{A}$ sent in step 2 and 3 cause an abort is the same as $\mathcal{S}$'s query to $\mathcal{F}_{\mathsf{LaAND}}$, noticing that the following is true.

$$\begin{aligned}
&\bigoplus_i \bigoplus_{j\neq i} x_i y^j_i \oplus \bigoplus_i x^i y^i \oplus \bigoplus_i (e^i \oplus r^i)\\
&= \bigoplus_i \bigoplus_{j\in\mathcal{M}, j\neq i} x_i y^j_i \oplus \bigoplus_i \bigoplus_{j\in\mathcal{H}, j\neq i} x_i y^j \oplus \bigoplus_i x^i y^i \oplus \bigoplus_i z^i \oplus \bigoplus_{i\in\mathcal{M}} q_i\\
&= \bigoplus_i \bigoplus_{j\in\mathcal{M}, j\neq i} x_i r_{j,i} \oplus \bigoplus_i \bigoplus_{j\neq i} x_i y^j \oplus \bigoplus_i z^i \oplus \bigoplus_{i\in\mathcal{M}} q_i\\
&= \bigoplus_i \bigoplus_{j\in\mathcal{M}, j\neq i} x_i r_{j,i} \oplus \bigoplus_{i\in\mathcal{M}} q_i\\
&= \left(\bigoplus_i x_i \bigoplus_{j\in\mathcal{M}, j\neq i} r_{j,i}\right) \oplus \bigoplus_{i\in\mathcal{M}} q_i
\end{aligned}$$

We will focus on the last step. If in step 6, it is the case that $\left(\bigoplus_i x^i\right)\left(\bigoplus_i y^i\right) \neq \left(\bigoplus_i z^i\right)$, then it is easy to see that the views are indistinguishable: all parties behave the same between hybrids. According to the unforgeability lemma, the protocol will abort with all but negligible probability. In the following, we will further focus on the case when the equation holds.

Note that in the idea world protocol, all H_i from $\mathcal{H}$ are picked randomly. We need to show that in the real world protocol all H_i's is also a random share of 0. In particular, we define

$$F^*_i = \left(\bigoplus_{k\neq i} \mathsf{K}_i[x^k]_{\Phi_i} \oplus \mathsf{M}_k[x^i]_{\Phi_k}\right)$$

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

Honest parties: For each $i \in [n]$, the box picks random $\langle x\rangle, \langle y\rangle, \langle z\rangle$ such that $(\bigoplus x^i) \wedge (\bigoplus y^i) = \bigoplus z^i$.

Corrupted parties: Corrupted parties get to choose all of their randomness.

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

Figure 17: Functionality $\mathcal{F}_{\mathsf{aAND}}$ for generating AND triples

and will first show that for any proper subset $S \subset \mathcal{H}$, $\bigoplus_{i\in S} F_i$ is indistinguishable from random to the $\mathcal{A}$. We use e to denote an honest party such that $e \in \mathcal{H}, e \notin S$. Such e always exists, since S is a proper subset of $\mathcal{H}$.

$$\begin{aligned}
\bigoplus_{i\in S} F_i^* &= \bigoplus_{i\in S}\bigoplus_{k\neq i}\left(\mathsf{K}_i[x^k]_{\Phi_i} \oplus \mathsf{M}_k[x^i]_{\Phi_k}\right)\\
&= \bigoplus_{i\in S}\bigoplus_{k\neq i}\left(\mathsf{K}_i[x^k]_{\Phi_i}\right) \oplus \bigoplus_{i\in S}\bigoplus_{k\neq i}\left(\mathsf{M}_k[x^i]_{\Phi_k}\right)\\
&= \bigoplus_{i\in S}\bigoplus_{k\neq i}\left(\mathsf{K}_i[x^k]_{\Phi_i}\right) \oplus \bigoplus_{k\in S}\bigoplus_{i\neq k}\left(\mathsf{M}_i[x^k]_{\Phi_i}\right)\\
&= \bigoplus_{i\in S}\bigoplus_{k\neq i}\left(\mathsf{K}_i[x^k]_{\Phi_i}\right) \oplus \bigoplus_{i\in[n]}\ \bigoplus_{k\in S, k\neq i}\left(\mathsf{M}_i[x^k]_{\Phi_i}\right)
\end{aligned}$$

From the equation, it is clear that for $i \in S$, $\mathsf{K}_e[x^i]$ is not in the computation, while $\mathsf{M}_e[x^i]$ is. Since $\mathsf{K}_e[x^i]$ is randomly picked by P_e, we know $\bigoplus_{i\in S} F_i^*$ is random. Therefore we can see that for any proper subset $S \subset \mathcal{H}$, $\bigoplus_{i\in S} H_i$ is indistinguishable from random.

Finally we need to show that the probability of abort due to selective failure attack is also the same. This is straightforward given the equation used in the unforgeability proof:

$$\begin{aligned}
\bigoplus_i H_i &= \bigoplus_{i\in\mathcal{M}} H_i \oplus \bigoplus_{i\in\mathcal{H}} H_i\\
&= \bigoplus_{i\in\mathcal{M}} (H_i^* \oplus Q_i) \oplus \bigoplus_{i\in\mathcal{H}}\left(H_i^* \oplus \left(\bigoplus_{k\neq i} x^k Q_{k,i}\right)\right)\\
&= \bigoplus_i H_i^* \oplus \bigoplus_{i\in\mathcal{M}} Q_i \oplus \bigoplus_{i\in\mathcal{H}}\left(\bigoplus_{k\neq i} x^k Q_{k,i}\right)\\
&= \bigoplus_{i\in\mathcal{M}} Q_i \oplus \bigoplus_{i\in\mathcal{H}}\left(\bigoplus_{k\neq i} x^k Q_{k,i}\right)\\
&= \bigoplus_{i\in\mathcal{M}} Q_i \oplus \bigoplus_k\left(x^k \bigoplus_{i\in\mathcal{H}, i\neq k} Q_{k,i}\right)\\
&= \bigoplus_{i\in\mathcal{M}} Q_i \oplus \bigoplus_k x^k R_k
\end{aligned}$$

□

B.5 Multi-Party Authenticated AND Triple

Once we have a protocol for leaky authenticated AND triple, it is straightforward to obtain a non-leaky authenticated AND triple, using the combine protocol in [WRK17]. We show the details of the protocol in Figure 18.

Protocol Π_{aAND}

1. P_i call $\mathcal{F}_{\mathsf{LaAND}}$ $\ell' = \ell B$ times and obtains $\{\langle x_j \rangle, \langle y_j \rangle, \langle z_j \rangle\}_{j \in [\ell']}$.
2. All parties randomly partition all objects into ℓ buckets, each with B objects.
3. For each bucket, parties combine B leaky ANDs into one non-leaky AND. To combine two leaky ANDs, namely $(\langle x_1 \rangle, \langle y_1 \rangle, \langle z_1 \rangle)$ and $(\langle x_2 \rangle, \langle y_2 \rangle, \langle z_2 \rangle)$:
   (a) Parties reveal $d := y_1 \oplus y_2$ with its MACs checked.
   (b) Each party P_i sets $\langle x \rangle := \langle x_1 \rangle \oplus \langle x_2 \rangle$, $\langle y \rangle := \langle y_1 \rangle$, $\langle z \rangle := \langle z_1 \rangle \oplus \langle z_2 \rangle \oplus d\langle x_2 \rangle$.

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

Figure 18: Protocol Π_{aAND} instantiating $\mathcal{F}_{\mathsf{aAND}}$.

Continent	Region	
North America	North Virginia	Ohio
	North California	Oregon
	Toronto	
Europe	Ireland	London
	Frankfurt	
Asia	Mumbai	Tokyo
	Seoul	Singapore
Australia	Sydney	
South America	São Paulo	

Table 7: List of all Amazon EC2 regions used in the WAN experiment.

C More Evaluation Results

In Table 8, we present all running time of basic circuit from 2 parties to 14 parties, in both the LAN setting and the WAN setting as described in Section 6.

Table 9 shows the experimental results for large number of parties the results are for a LAN setting.

Table 11 shows details of the communication of our implementation. All numbers are counted from our implementation and thus slightly higher than theoretically the best possible. All numbers are the maximum amount of communication for a party.

		LAN					WAN				
n		Setup	Indep.	Depen.	Online	Total	Setup	Indep.	Depen.	Online	Total
2	AES	33.5	37.4	7.6	1.3	79.8	43.7	286.7	32.5	12.1	375.0
	SHA-1	33.8	148.2	25.2	5.0	212.2	43.9	587.0	101.7	15.9	748.5
	SHA-256	34.3	329.3	58.7	11.8	434.1	44.3	1097.0	205.2	23.1	1369.6
3	AES	35.2	46.7	11.5	2.0	95.4	213.6	1455.1	183.8	66.0	1918.5
	SHA-1	35.5	203.6	36.2	9.3	284.6	212.5	2038.4	270.7	74.7	2596.3
	SHA-256	36.7	470.1	88.0	23.4	618.2	213.7	2939.7	511.0	87.8	3752.2
4	AES	36.5	57.7	15.7	3.2	113.1	321.2	1903.3	319.2	85.4	2629.1
	SHA-1	36.8	265.5	50.3	13.7	366.3	322.7	2665.4	358.7	96.6	3443.4
	SHA-256	37.9	632.2	123.7	31.6	825.4	328.2	3669.0	786.5	113.1	4896.8
5	AES	38.4	72.4	19.9	4.3	135.0	727.8	1988.6	327.6	82.7	3126.7
	SHA-1	39.2	326.2	73.9	20.8	460.1	719.0	2828.3	442.5	106.0	4095.8
	SHA-256	40.0	792.3	182.7	48.9	1063.9	728.4	3953.5	1101.1	131.2	5914.2
6	AES	42.8	85.3	26.1	5.4	159.6	1151.8	3432.1	511.2	87.8	5182.9
	SHA-1	44.5	410.8	110.3	26.1	591.7	1131.1	4921.8	1159.2	118.6	7330.7
	SHA-256	46.0	968.2	282.0	63.8	1360.0	1112.3	6933.2	1617.7	149.8	9813.0
7	AES	47.0	99.8	31.8	7.3	185.9	1459.5	3986.6	620.3	90.8	6157.2
	SHA-1	49.7	493.1	141.8	31.2	715.8	1480.4	5690.3	936.0	115.4	8222.1
	SHA-256	49.4	1176.3	397.0	74.5	1697.2	1455.3	7862.8	1765.8	155.7	11239.6
8	AES	50.9	114.7	38.7	8.5	212.8	1767.7	4189.7	710.6	100.4	6768.4
	SHA-1	52.4	551.2	180.1	39.9	823.6	1791.9	5920.0	1065.8	132.7	8910.4
	SHA-256	51.4	1328.0	470.7	94.5	1944.6	1807.1	8504.8	2129.4	185.0	12626.3
9	AES	53.4	134.7	45.2	9.7	243.0	3279.2	6237.3	1356.2	198.3	11071.0
	SHA-1	54.4	613.6	236.7	47.2	951.9	3263.7	8843.3	2769.5	246.9	15123.4
	SHA-256	54.7	1505.4	573.5	114.9	2248.5	3274.6	12712.0	4705.4	315.2	21007.2
10	AES	59.3	144.2	53.0	11.7	268.2	4502.8	6279.1	1388.2	200.7	12370.8
	SHA-1	60.3	721.0	285.2	54.9	1121.4	4479.5	8962.9	2671.9	252.4	16366.7
	SHA-256	60.2	1694.9	658.3	128.4	2541.8	4524.3	12465.7	6037.4	328.9	23356.3
11	AES	61.3	165.8	60.9	15.0	303.0	4709.8	7083.2	1534.9	205.0	13532.9
	SHA-1	63.0	798.3	318.1	69.8	1249.2	4823.6	12112.6	3539.2	273.7	20749.1
	SHA-256	62.1	1904.9	881.1	173.2	3021.3	4752.5	13432.2	5501.9	378.4	24065.0
12	AES	68.0	181.6	75.8	16.7	342.1	4860.2	7349.1	1718.9	232.1	14160.3
	SHA-1	71.4	872.8	382.4	79.3	1405.9	4826.6	10596.2	3519.9	302.8	19245.5
	SHA-256	69.7	2045.5	1033.9	200.5	3349.6	4814.5	16738.7	6743.5	437.9	28734.6
13	AES	73.5	237.2	85.6	18.4	414.7	6703.6	8531.2	1932.3	248.4	17415.5
	SHA-1	74.0	1281.8	465.1	88.8	1909.7	6743.2	12151.9	4051.9	324.5	23271.5
	SHA-256	75.5	2953.4	1277.7	216.5	4523.1	6682.9	18355.0	8879.7	490.8	34408.4
14	AES	76.8	258.2	102.5	20.3	457.8	8710.8	9412.2	1947.0	250.2	20320.2
	SHA-1	77.0	1375.3	546.7	91.2	2090.2	8654.9	13512.7	4118.9	338.7	26625.2
	SHA-256	79.7	3283.2	1573.0	238.8	5174.7	8714.4	18104.8	8236.1	480.2	35535.5

Table 8: Detailed numbers for experiments of basic circuits. Timings are measured in terms of milliseconds.

n	Setup	Indep.	Depen.	Online	Total
2	33.0	36.3	7.0	1.3	77.5
4	36.5	54.6	15.0	3.1	109.2
8	49.4	122.2	35.7	7.8	215.1
16	78.8	227.8	121.1	29.4	457.0
32	129.9	627.0	446.2	112.3	1315.5
64	212.9	1182.2	2630.1	476.5	4501.7
128	383.0	2727.4	11669.6	1870.2	16650.2

Table 9: Detailed experiment results for the crowd setting in the LAN. Timings are measured in terms of milliseconds.

n	Setup	Indep.	Depen.	Online	Total
8	16736.0	30647.4	5905.2	783.3	54071.9
16	18708.1	21699.6	12243.5	598.8	53250.0
32	35838.8	19038.4	8242.3	716.6	63836.1
64	71913.8	25280.7	29416.6	1564.0	128175.2
128	88055.9	30795.9	22659.1	2316.2	143827.0

Table 10: Detailed experiment results for the worldwide crowd setting. Timings are measured in terms of milliseconds.

n		Setup	Indep.	Depen.	Online	Total
	AES	28.6 KB	2.4 MB	0.9 MB	4.5 KB	3.3 MB
2	SHA-1	28.6 KB	13.0 MB	4.7 MB	8.9 KB	17.8 MB
	SHA-256	28.6 KB	31.7 MB	11.4 MB	9.0 KB	43.1 MB
	AES	57.1 KB	4.8 MB	1.3 MB	4.5 KB	6.2 MB
3	SHA-1	57.1 KB	26.1 MB	7.2 MB	8.9 KB	33.3 MB
	SHA-256	57.1 KB	63.3 MB	17.4 MB	9.0 KB	80.8 MB
	AES	85.7 KB	7.2 MB	1.8 MB	4.5 KB	9.1 MB
4	SHA-1	85.7 KB	39.1 MB	9.6 MB	8.9 KB	48.9 MB
	SHA-256	85.7 KB	95.0 MB	23.4 MB	9.0 KB	118.5 MB
	AES	114.2 KB	9.7 MB	2.2 MB	4.5 KB	12.0 MB
5	SHA-1	114.2 KB	52.2 MB	12.1 MB	8.9 KB	64.4 MB
	SHA-256	114.2 KB	126.6 MB	29.4 MB	9.0 KB	156.2 MB
	AES	142.8 KB	12.1 MB	2.7 MB	4.5 KB	14.9 MB
6	SHA-1	142.8 KB	65.2 MB	14.5 MB	8.9 KB	79.9 MB
	SHA-256	142.8 KB	158.3 MB	35.4 MB	9.0 KB	193.9 MB
	AES	171.4 KB	14.5 MB	3.1 MB	4.5 KB	17.8 MB
7	SHA-1	171.4 KB	78.3 MB	17.0 MB	8.9 KB	95.5 MB
	SHA-256	171.4 KB	190.0 MB	41.4 MB	9.0 KB	231.6 MB
	AES	199.9 KB	16.9 MB	3.5 MB	4.5 KB	20.7 MB
8	SHA-1	199.9 KB	91.3 MB	19.5 MB	8.9 KB	111.0 MB
	SHA-256	199.9 KB	221.7 MB	47.4 MB	9.0 KB	269.3 MB
	AES	428.4 KB	36.4 MB	7.1 MB	4.5 KB	44.0 MB
16	SHA-1	428.4 KB	195.9 MB	39.2 MB	8.9 KB	235.5 MB
	SHA-256	428.4 KB	475.1 MB	95.4 MB	9.0 KB	570.9 MB

Table 11: Communication complexity of our protocol. Bandwidth are measured in terms of megabytes.