Authenticated Garbling and Efficient Maliciously Secure Multi-Party Computation

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

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

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

Abstract

In this paper, we extend the recent work by Wang et al., who proposed a new framework for secure two-party computation in the preprocessing model that can be instantiated efficiently using TinyOT. We show that their protocol can be generalized to the multi-party setting, where the preprocessing functionality is based on the multi-party TinyOT-like protocol. Assuming there are n parties where at most $n-1$ parties are corrupted, the function-dependent phase has a total communication complexity of $O(\kappa n^2)$ bits per AND gate; the online phase has a total communication complexity of $O(\kappa n^2)$ bits per input/output bit.

In the second part of this paper, we propose a new multi-party TinyOT protocol. The new protocol uses a set of new techniques that allow parties to distributively check the correctness without the need for cut-and-choose. The resulting protocol is much more efficient compared to previous protocols: with statistical security parameter ρ, the complexity to generate one AND triple is $O(\frac{\rho}{\log|\mathcal{C}|}n^2)$, where $|\mathcal{C}|$ is the circuit size. The best previous multi-party TinyOT protocol by Frederiksen et al. has a complexity of $O(\frac{\rho^2}{\log^2|\mathcal{C}|}n^2)$ per AND triple. The complexity is measured in terms of number of symmetric key operations/number of symmetric key messages.

The resulting protocol enjoys extremely high efficiency, compared to the state-of-the-art protocol by Lindell et al. that combines the BMR protocol with the SPDZ protocol.

1 Introduction

Secure multi-party computation (MPC) allows a set of parties to privately compute a function on their joint inputs. The protocol ensures that an adversary corrupting a set of parties cannot learn anything more than the output of the function.

Most works on MPC [KOS16, FKOS15, BDOZ11, DPSZ12, FLNW17] suffer from one huge drawback: the protocol requires a number of round-trips proportional to the depth of the circuit being evaluated. When the parties are geographically separated, or when the number of parties becomes high, such round-trip time can be overwhelming, compared to the time required for the other parts of the protocol. For example, the AES circuit has a depth of around 50. If parties are located in the U.S. and Europe, round-trip time is 75 ms even with dedicated networks provided by Amazon EC2. This means a total of 3750 ms is spent on round-trips, not to mentioned the time to perform cryptographic operations and to send messages. Most of these works are based on secret sharing: for each AND/Mult gate, they require at least one round-trip.

Another completely different approach that constructs constant-round MPC protocol is by Beaver, Micali, and Rogaway [BMR90]. Their protocol uses any interactive MPC protocol to jointly

	Complexity
Choi et al. [CKMZ14]	$O\left(\lvert\mathcal{C}\rvert\frac{\rho^2}{\log\lvert\mathcal{C}\rvert}\right)$
Lindell et al. [LPSY15] + Keller et al. [KOS16]	$O\left(\lvert\mathcal{C}\rvert\kappa n^2\right)$
Section 4 + Frederiksen et al. [FKOS15]	$O\left(\lvert\mathcal{C}\rvert\frac{\rho^2}{\log^2\lvert\mathcal{C}\rvert}n^2\right)$
Section 4 + Section 6	$O\left(\lvert\mathcal{C}\rvert\frac{\rho}{\log\lvert\mathcal{C}\rvert}n^2\right)$

Table 1: **Constant-round Multi-party protocol secure against $n-1$ corruption.** Choi et al. is only for three party.

garble a circuit, which can be then evaluated. This approach was regarded as only a theoretical solution until recently: Damgård and Ishai [DI05] applied this idea to a setting with honest majority; Choi et al. [CKMZ14] applied to a setting with three parties and dishonest majority; Lindell et al. [LPSY15, LSS16] uses the SPDZ protocol and Somewhat Homomorphic Encryption to garble a circuit achieving constant round MPC protocol with all-but-one corruption.

Authenticated Garbling. A recent work by Wang et al. shows that in the two-party setting, BMR protocol can be made practical. In particular, the paper discusses two techniques: 1) how to use the TinyOT protocol to distributively garble a single circuit efficiently. The distributed garbled circuit is "authenticated" such that an adversary cannot arbitrarily change it; 2) how to use ideas from zero-knowledge garbled circuit [JKO13] to, in turn, construct an improved TinyOT protocol.

Contribution. In this paper, we fully extend the authenticated garbling in the multi-party setting:

1. We present an extension of the main protocol by Wang et al. to the multi-party setting. The resulting protocol is in the (TinyOT-like) preprocessing model, with communication $O(\kappa n^2|\mathcal{C}|)$ bits and only has a constant number of rounds.

2. The above preprocessing functionality can be instantiated using existing work. However, we design a new protocol that generalizes TinyOT for the multi-party setting. The cost to preprocess a single AND gate is $O(Bn^2)$ where $B = \frac{\rho}{\log|\mathcal{C}|}$, while the best previous work [FKOS15] requires $O(B^2n^2)$.

3. The resulting protocol is very simple, and we intend to implement it to test its practical performance.

Outline. In the next section, we will provide some high-level intuition on how our main protocol in the preprocessing model works. In Section 4, we provide complete description of the protocol, with proof in Section 5. Finally in Section 6, we discuss an efficient instantiation of the preprocessing functionality.

2 Notations and Preliminaries

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

We represent a circuit as a list of gates. Each gate is represented as $(\alpha, \beta, \gamma, T)$, which means a gate with input-wire indices as α and β; output wire index as γ and gate type as $T \in \{\oplus, \wedge\}$. Furthermore, 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, $\mathcal{O}$ to denote the set of output wire indices of the circuit. Parties are denoted as $P_1, ..., P_n$. Since our main protocol is based on garbled circuits, we designate P_1 as the circuit evaluator. $\mathcal{M}$ is used to denote the set of parties that are corrupted and $\mathcal{H}$ is used to denote the set of honest parties, which means $\mathcal{M} \cup \mathcal{H} = [n]$.

Information-theoretic MAC (IT-MAC). We use a multi-party variant of the information-theoretic message authentication code originally used by Nielsen et al. [NNOB12]. We follow the description from Wang et al. [WRK17]. Each player holds a global key Δ_i. To allow the player P_i to hold a MAC tag for the value b towards player P_j, we give a κ-bit long random key $\mathsf{K}_j[b]$ to P_j and give $\mathsf{M}_j[b] := \mathsf{K}_j[b] \oplus b\Delta_j$ to player P_i. We also allow a player P_i to authenticate a single value x towards all other players: for each $j \in [n], j \neq i$, we give, P_j a random key $\mathsf{K}_j[x]$ and give $\mathsf{M}_j[b] := \mathsf{K}_j[b] \oplus b\Delta_j$ to P_i. This is equivalent to authenticating the same bit to all other parties. (We use $\mathcal{F}^n_{\mathsf{aBit}}$ to model this as an ideal functionality and discuss an efficient instantiation in Section 6.1.)

We will use $[x]^i$ to denote a multi-party IT-MAC for a bit x held by P_i. $[x]^i$ therefore means $(x, \{\mathsf{M}_k[x]\}_{k \neq i})$ for P_i, and $[x]^i$ means $\mathsf{K}_j[x]$ for P_j with $j \neq i$. Note that $[x]^i$ is XOR-homomorphic: given two authenticated bits $[x]^i, [y]^i$, it is possible to generate an authenticated bit $[z]^i$ whose value is the XOR of the two authenticated bits by doing the following:

1. $z := x \oplus y$
2. $\mathsf{K}_j[z] := \mathsf{K}_j[x] \oplus \mathsf{K}_j[y], \forall j \neq i$
3. $\mathsf{M}_j[z] := \mathsf{M}_j[x] \oplus \mathsf{M}_j[y], \forall j \neq i$

It is also possible to negate $[x]^i$ resulting in $[y]^i$:

1. $y := x \oplus 1$
2. $\mathsf{K}_j[y] := \mathsf{K}_j[x] \oplus \Delta_j, \forall j \neq i$
3. $\mathsf{M}_j[y] := \mathsf{M}_j[x], \forall j \neq i$

In the above construction, x is known to one party. To generate a distributed authenticated bit x, where the value is not known to any party, we generate shares for x, namely $\bigoplus_i x^i = x$. For each x^i, parties also obtain $[x^i]^i$, that is, multi-party MACs on x^i with P_i holding x^i. We will use $\langle x \rangle$ to denote authenticated shares for bit x, that is $\langle x \rangle = \{[x^i]^i\}^{i \in [n]}$.

Note that the above representation assumes that the global key are Δ's. In the case where global keys are some G's, we explicitly add a subscript to the representation: we use $\mathsf{M}_i[x]_{G_i}$, $\mathsf{K}_i[x]_{G_i}$ to represent the MAC and keys. That is $\mathsf{M}_i[x]_{G_i} = \mathsf{K}_i[x]_{G_i} \oplus xG_i$.

Related functionalities. The MPC functionality that our main protocol instantiates is shown in Figure 1. We focus on a simplified version where only P_1 gets the output. Our main protocol works in the $\mathcal{F}_{\mathsf{Pre}}$-hybrid model. The detailed ideal functionality $\mathcal{F}_{\mathsf{Pre}}$ is shown in Figure 2. From a high level view, $\mathcal{F}_{\mathsf{Pre}}$ generates multi-party IT-MAC on some value x, y, z such that $z = x \wedge y$. In the later section, we also refer to this set of multi-party IT-MACs as a AND triple.

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

Private inputs: P_i has input $x^i \in \{0,1\}^{n_i}$.

1. Upon receiving (input, x^i) from P_i, store the message (i, x^i) if no message of the form $(i, \cdot)$ is present in memory. If (i, x^i) is present in memory for all $i \in [n]$, the box computes $z := f(x_1, ..., x_n)$ and sends z to P_1.

Figure 1: Functionality $\mathcal{F}_{\mathsf{mpc}}$ for multi-party computation.

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

1. Upon receiving init from all P_i's, sample $\{\Delta_i \in \{0,1\}^\kappa\}_{i\in[n]}$ and send Δ_i to P_i. Corrupted parties can choose their own Δ_i.
2. Upon receiving random from all P_i, sample a random bit r and a random multi-party IT-MAC, namely $\{(r_i, \{\mathsf{M}_j[r_i], \mathsf{K}_j[r_i]\}_{j\neq i})\}_{i\in[n]}$. The box sends $(r_i, \{\mathsf{M}_j[r_i], \mathsf{K}_i[r_j]\}_{j\neq i})$ to P_i. Corrupted parties can choose their own randomness.
3. Upon receiving $(\mathtt{AND}, (r_i, \{\mathsf{M}_j[r_i], \mathsf{K}_i[r_j]\}_{j\neq i}), (s_i, \{\mathsf{M}_j[s_i], \mathsf{K}_i[s_j]\}_{j\neq i})$ from P_i, the box checks that each IT-MAC is valid, picks random $\{t_i\}_{i\in[n]}$, such that $\bigoplus_{i\in[n]} t_i = (\bigoplus_{i\in[n]} r_i) \wedge (\bigoplus_{i\in[n]} s_i)$ and random IT-MACs $\{(t_i, \{\mathsf{M}_j[t_i], \mathsf{K}_j[t_i]\}_{j\neq i})\}_{i\in[n]}$ on them, and sends $(r_i, \{\mathsf{M}_j[r_i], \mathsf{K}_i[r_j]\}_{j\neq i})$ to P_i. Corrupted party gets to choose its own randomness.

Figure 2: The multi-party preprocessing Functionality.

3 Protocol Intuition

Our main protocol can be viewed as a (non-trivial) extension of a recent work by Wang et al. that proposed an authenticated garbling protocol for maliciously-secure two-party computation. From a high-level view, their protocol constructs shares of a garbled table where permutation bits are authenticated in the following way.

$x \oplus \lambda_\alpha$	$y \oplus \lambda_\beta$	P_2's share of Garbled Table	P_1's share of Garbled Table
0	0	$H(\mathsf{L}_{\alpha,0}, \mathsf{L}_{\beta,0}, \gamma, 00) \oplus (r^2_{00}, \mathsf{M}_1[r^2_{00}], R_{00} \oplus \mathsf{L}_{\gamma,\bar{z}_{00}})$	$(r^1_{00} = \bar{z}_{00} \oplus r^2_{00}, \mathsf{K}_1[r^2_{00}], R_{00})$
0	1	$H(\mathsf{L}_{\alpha,0}, \mathsf{L}_{\beta,1}, \gamma, 01) \oplus (r^2_{01}, \mathsf{M}_1[r^2_{01}], R_{01} \oplus \mathsf{L}_{\gamma,\bar{z}_{01}})$	$(r^1_{01} = \bar{z}_{01} \oplus r^2_{01}, \mathsf{K}_1[r^2_{10}], R_{01})$
1	0	$H(\mathsf{L}_{\alpha,1}, \mathsf{L}_{\beta,0}, \gamma, 10) \oplus (r^2_{10}, \mathsf{M}_1[r^2_{10}], R_{10} \oplus \mathsf{L}_{\gamma,\bar{z}_{10}})$	$(r^1_{10} = \bar{z}_{10} \oplus r^2_{10}, \mathsf{K}_1[r^2_{01}], R_{10})$
1	1	$H(\mathsf{L}_{\alpha,1}, \mathsf{L}_{\beta,1}, \gamma, 11) \oplus (r^2_{11}, \mathsf{M}_1[r^2_{11}], R_{11} \oplus \mathsf{L}_{\gamma,\bar{z}_{11}})$	$(r^1_{11} = \bar{z}_{11} \oplus r^2_{11}, \mathsf{K}_1[r^2_{11}], R_{11})$

With an appropriate choice of the preprocessing functionality, each row of the garbled table can easily be computed as follows.

$x \oplus \lambda_\alpha$	$y \oplus \lambda_\beta$	P_1's share of Garbled Table	P_2's share of Garbled Table
0	0	$H(\mathsf{L}_{\alpha,0}, \mathsf{L}_{\beta,0}, \gamma, 00) \oplus (r_{00}, \mathsf{M}[r_{00}], L_{\gamma,0} \oplus r_{00}\Delta_1 \oplus \mathsf{K}[s_{00}])$	$(s_{00} = \bar{z}_{00} \oplus r_{00}, \mathsf{K}[r_{00}], \mathsf{M}[s_{00}])$
0	1	$H(\mathsf{L}_{\alpha,0}, \mathsf{L}_{\beta,1}, \gamma, 01) \oplus (r_{01}, \mathsf{M}[r_{01}], L_{\gamma,0} \oplus r_{01}\Delta_1 \oplus \mathsf{K}[s_{01}])$	$(s_{01} = \bar{z}_{01} \oplus r_{01}, \mathsf{K}[r_{01}], \mathsf{M}[s_{01}])$
1	0	$H(\mathsf{L}_{\alpha,1}, \mathsf{L}_{\beta,0}, \gamma, 10) \oplus (r_{10}, \mathsf{M}[r_{10}], L_{\gamma,0} \oplus r_{10}\Delta_1 \oplus \mathsf{K}[s_{10}])$	$(s_{10} = \bar{z}_{10} \oplus r_{10}, \mathsf{K}[r_{10}], \mathsf{M}[s_{10}])$
1	1	$H(\mathsf{L}_{\alpha,1}, \mathsf{L}_{\beta,1}, \gamma, 11) \oplus (r_{11}, \mathsf{M}[r_{11}], L_{\gamma,0} \oplus r_{11}\Delta_1 \oplus \mathsf{K}[s_{11}])$	$(s_{11} = \bar{z}_{11} \oplus r_{11}, \mathsf{K}[r_{11}], \mathsf{M}[s_{11}])$

Our main goal is to generalize these ideas to $n > 2$ parties where $n-1$ parties jointly garble and then let the excluded party evaluate the garbled circuit. Since up to $n-1$ parties can be malicious, the garbled circuit and associated permutation bits need to be shared among all parties such that no subset of the parties can recover the garbled circuit. Similar to the original 2PC protocol, permutation bits also need to be authenticated.

In the following example, we will restrict ourselves to the three-party setting. The first step is to extend the garbled table without considering how to construct the authenticated garbling.

P_3's share of Garbled Table	P_2's share of Garbled Table
$H(\mathsf{L}^3_{\alpha,0}, \mathsf{L}^3_{\beta,0}, \gamma, 00) \oplus (r^3_{00}, \mathsf{M}_1[r^3_{00}], R^2_{00} \oplus R^1_{00} \oplus \mathsf{L}^3_{\gamma,\bar{z}_{00}}, S^3_{00})$	$H(\mathsf{L}^2_{\alpha,0}, \mathsf{L}^2_{\beta,0}, \gamma, 00) \oplus (r^2_{00}, \mathsf{M}_1[r^2_{00}], S^3_{00} \oplus S^1_{00} \oplus \mathsf{L}^2_{\gamma,\bar{z}_{00}}, R^2_{00})$
$H(\mathsf{L}^3_{\alpha,0}, \mathsf{L}^3_{\beta,1}, \gamma, 01) \oplus (r^3_{01}, \mathsf{M}_1[r^3_{01}], R^2_{01} \oplus R^1_{01} \oplus \mathsf{L}^3_{\gamma,\bar{z}_{01}}, S^3_{01})$	$H(\mathsf{L}^2_{\alpha,0}, \mathsf{L}^2_{\beta,1}, \gamma, 01) \oplus (r^2_{01}, \mathsf{M}_1[r^2_{01}], S^3_{01} \oplus S^1_{01} \oplus \mathsf{L}^2_{\gamma,\bar{z}_{01}}, R^2_{01})$
$H(\mathsf{L}^3_{\alpha,1}, \mathsf{L}^3_{\beta,0}, \gamma, 10) \oplus (r^3_{10}, \mathsf{M}_1[r^3_{10}], R^2_{10} \oplus R^1_{10} \oplus \mathsf{L}^3_{\gamma,\bar{z}_{10}}, S^3_{10})$	$H(\mathsf{L}^2_{\alpha,1}, \mathsf{L}^2_{\beta,0}, \gamma, 10) \oplus (r^2_{10}, \mathsf{M}_1[r^2_{10}], S^3_{10} \oplus S^1_{10} \oplus \mathsf{L}^2_{\gamma,\bar{z}_{10}}, R^2_{10})$
$H(\mathsf{L}^3_{\alpha,1}, \mathsf{L}^3_{\beta,1}, \gamma, 11) \oplus (r^3_{11}, \mathsf{M}_1[r^3_{11}], R^2_{11} \oplus R^1_{11} \oplus \mathsf{L}^3_{\gamma,\bar{z}_{11}}, S^3_{11})$	$H(\mathsf{L}^2_{\alpha,1}, \mathsf{L}^2_{\beta,1}, \gamma, 11) \oplus (r^2_{11}, \mathsf{M}_1[r^2_{11}], S^3_{11} \oplus S^1_{11} \oplus \mathsf{L}^2_{\gamma,\bar{z}_{11}}, R^2_{11})$

P_1's share of Garbled Table
$(r^1_{00} = \bar{z}_{00} \oplus r^3_{00} \oplus r^2_{00}, \mathsf{K}_1[r^3_{00}], \mathsf{K}_1[r^2_{00}], R^1_{00}, S^1_{00})$
$(r^1_{01} = \bar{z}_{01} \oplus r^3_{01} \oplus r^2_{01}, \mathsf{K}_1[r^3_{01}], \mathsf{K}_1[r^2_{01}], R^1_{01}, S^1_{01})$
$(r^1_{10} = \bar{z}_{10} \oplus r^3_{10} \oplus r^2_{10}, \mathsf{K}_1[r^3_{10}], \mathsf{K}_1[r^2_{10}], R^1_{10}, S^1_{10})$
$(r^1_{11} = \bar{z}_{11} \oplus r^3_{11} \oplus r^2_{11}, \mathsf{K}_1[r^3_{11}], \mathsf{K}_1[r^2_{11}], R^1_{11}, S^1_{11})$

In the above example, r^i_{uv}'s are random bits while R^i_{uv}, S^i_{uv} are random κ-bit strings. As can be noticed, there are two sets of garbled labels used by P_2 and P_3 respectively. Furthermore, both of these sets are shared among the three parties such that the garbled circuit remains private even when the adversary corrupts all-but-one parties. The last ingredient that we need is a protocol for constructing these "shared and permuted garbled labels" distributively. Observe that

$$\begin{aligned}
\mathsf{L}^3_{\gamma,\bar{z}_{00}} &= \mathsf{L}^3_{\gamma,0} \oplus \bar{z}_{00}\Delta_3 \\
&= \mathsf{L}^3_{\gamma,0} \oplus (r^1_{00} \oplus r^2_{00} \oplus r^3_{00})\Delta_3 \\
&= (\mathsf{L}^3_{\gamma,0} \oplus r^3_{00}\Delta_3 \oplus \mathsf{K}_3[r^1_{00}] \oplus \mathsf{K}_3[r^2_{00}]) \oplus (\mathsf{K}_3[r^1_{00}] \oplus r^1_{00}\Delta_3) \oplus (\mathsf{K}_3[r^2_{00}] \oplus r^2_{00}\Delta_3) \\
&= (\mathsf{L}^3_{\gamma,0} \oplus r^3_{00}\Delta_3 \oplus \mathsf{K}_3[r^1_{00}] \oplus \mathsf{K}_3[r^2_{00}]) \oplus \mathsf{M}_3[r^1_{00}] \oplus \mathsf{M}_3[r^2_{00}]
\end{aligned}$$

Applying this to the construction above, with the $\mathcal{F}_{\mathsf{Pre}}$ functionality described in Section 2:

P_3's share of Garbled Table
$H(\mathsf{L}^3_{\alpha,0}, \mathsf{L}^3_{\beta,0}, \gamma, 00) \oplus (r^3_{00}, \mathsf{M}_1[r^3_{00}], \mathsf{L}^3_{\gamma,0} \oplus r^3_{00}\Delta_3 \oplus \mathsf{K}_3[r^1_{00}] \oplus \mathsf{K}_3[r^2_{00}], \mathsf{M}_2[r^3_{00}])$
$H(\mathsf{L}^3_{\alpha,0}, \mathsf{L}^3_{\beta,1}, \gamma, 01) \oplus (r^3_{01}, \mathsf{M}_1[r^3_{01}], \mathsf{L}^3_{\gamma,0} \oplus r^3_{10}\Delta_3 \oplus \mathsf{K}_3[r^1_{10}] \oplus \mathsf{K}_3[r^2_{10}], \mathsf{M}_2[r^3_{01}])$
$H(\mathsf{L}^3_{\alpha,1}, \mathsf{L}^3_{\beta,0}, \gamma, 10) \oplus (r^3_{10}, \mathsf{M}_1[r^3_{10}], \mathsf{L}^3_{\gamma,0} \oplus r^3_{01}\Delta_3 \oplus \mathsf{K}_3[r^1_{01}] \oplus \mathsf{K}_3[r^2_{01}], \mathsf{M}_2[r^3_{10}])$
$H(\mathsf{L}^3_{\alpha,1}, \mathsf{L}^3_{\beta,1}, \gamma, 11) \oplus (r^3_{11}, \mathsf{M}_1[r^3_{11}], \mathsf{L}^3_{\gamma,0} \oplus r^3_{11}\Delta_3 \oplus \mathsf{K}_3[r^1_{11}] \oplus \mathsf{K}_3[r^2_{11}], \mathsf{M}_2[r^3_{11}])$
P_2's share of Garbled Table
$H(\mathsf{L}^2_{\alpha,0}, \mathsf{L}^2_{\beta,0}, \gamma, 00) \oplus (r^2_{00}, \mathsf{M}_1[r^2_{00}], \mathsf{L}^2_{\gamma,0} \oplus r^2_{00}\Delta_2 \oplus \mathsf{K}_2[r^1_{00}] \oplus \mathsf{K}_2[r^3_{00}], \mathsf{M}_3[r^2_{00}])$
$H(\mathsf{L}^2_{\alpha,0}, \mathsf{L}^2_{\beta,1}, \gamma, 01) \oplus (r^2_{01}, \mathsf{M}_1[r^2_{01}], \mathsf{L}^2_{\gamma,0} \oplus r^2_{10}\Delta_2 \oplus \mathsf{K}_2[r^1_{10}] \oplus \mathsf{K}_2[r^3_{10}], \mathsf{M}_3[r^2_{01}])$
$H(\mathsf{L}^2_{\alpha,1}, \mathsf{L}^2_{\beta,0}, \gamma, 10) \oplus (r^2_{10}, \mathsf{M}_1[r^2_{10}], \mathsf{L}^2_{\gamma,0} \oplus r^2_{01}\Delta_2 \oplus \mathsf{K}_2[r^1_{01}] \oplus \mathsf{K}_2[r^3_{01}], \mathsf{M}_3[r^2_{10}])$
$H(\mathsf{L}^2_{\alpha,1}, \mathsf{L}^2_{\beta,1}, \gamma, 11) \oplus (r^2_{11}, \mathsf{M}_1[r^2_{11}], \mathsf{L}^2_{\gamma,0} \oplus r^2_{11}\Delta_2 \oplus \mathsf{K}_2[r^1_{11}] \oplus \mathsf{K}_2[r^3_{11}], \mathsf{M}_3[r^2_{11}])$

P_1's share of Garbled Table
$(r^1_{00} = \bar{z}_{00} \oplus r^3_{00} \oplus r^2_{00}, \mathsf{K}_1[r^3_{00}], \mathsf{K}_1[r^2_{00}], \mathsf{M}_3[r^1_{00}], \mathsf{M}_2[r^1_{00}])$
$(r^1_{01} = \bar{z}_{01} \oplus r^3_{01} \oplus r^2_{01}, \mathsf{K}_1[r^3_{01}], \mathsf{K}_1[r^2_{01}], \mathsf{M}_3[r^1_{01}], \mathsf{M}_2[r^1_{01}])$
$(r^1_{10} = \bar{z}_{10} \oplus r^3_{10} \oplus r^2_{10}, \mathsf{K}_1[r^3_{10}], \mathsf{K}_1[r^2_{10}], \mathsf{M}_3[r^1_{10}], \mathsf{M}_2[r^1_{10}])$
$(r^1_{11} = \bar{z}_{11} \oplus r^3_{11} \oplus r^2_{11}, \mathsf{K}_1[r^3_{11}], \mathsf{K}_1[r^2_{11}], \mathsf{M}_3[r^1_{11}], \mathsf{M}_2[r^1_{11}])$

4 The Main Scheme

In Figure 3 and Figure 4, we present the complete MPC protocol in the $\mathcal{F}_{\mathsf{Pre}}$-hybrid model. In Section 6, we will introduce an efficient instantiation of $\mathcal{F}_{\mathsf{Pre}}$, which extends two-party TinyOT protocol. Note that similar to [NNOB12], the preprocessing functionality needs a global key query instruction. This does not affect the security for PPT adversaries.

5 Proof

Theorem 5.1. *The protocol in Figure 3 and Figure 4, 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 $\mathcal{F}_{\mathsf{mpc}}$.

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 5.1, 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.

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 $\left(r^i_w, \left\{\mathsf{M}_j[r^i_w], \mathsf{K}_i[r^j_w]\right\}_{j \neq i}\right)$ to P_i, where $\bigoplus_{i\in[n]} r^i_w = \lambda_w$. For each $i \neq 1$, P_i also picks a random κ-bit string $\mathsf{L}^i_{w,0}$.

Function-dependent phase:

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

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

   (b) For each $i \neq 1$, P_i computes the following locally.

$$\begin{aligned}
\left(r^i_{\gamma,0}, \left\{\mathsf{M}_j[r^i_{\gamma,0}], \mathsf{K}_i[r^j_{\gamma,0}]\right\}_{j\neq i}\right) &:= \left(r^i_\sigma \oplus r^i_\gamma, \quad \left\{\mathsf{M}_j[r^i_\sigma] \oplus \mathsf{M}_j[r^i_\gamma], \quad \mathsf{K}_i[r^j_\sigma] \oplus \mathsf{K}_i[r^j_\gamma]\right\}_{j\neq i}\right)\\
\left(r^i_{\gamma,1}, \left\{\mathsf{M}_j[r^i_{\gamma,1}], \mathsf{K}_i[r^j_{\gamma,1}]\right\}_{j\neq i}\right) &:= \left(r^i_{\gamma,0} \oplus r^i_\alpha, \left\{\mathsf{M}_j[r^i_{\gamma,0}] \oplus \mathsf{M}_j[r^i_\alpha], \quad \mathsf{K}_i[r^j_{\gamma,0}] \oplus \mathsf{K}_i[r^j_\alpha]\right\}_{j\neq i}\right)\\
\left(r^i_{\gamma,2}, \left\{\mathsf{M}_j[r^i_{\gamma,2}], \mathsf{K}_i[r^j_{\gamma,2}]\right\}_{j\neq i}\right) &:= \left(r^i_{\gamma,0} \oplus r^i_\beta, \left\{\mathsf{M}_j[r^i_{\gamma,0}] \oplus \mathsf{M}_j[r^i_\beta], \quad \mathsf{K}_i[r^j_{\gamma,0}] \oplus \mathsf{K}_i[r^j_\beta]\right\}_{j\neq i}\right)\\
\left(r^i_{\gamma,3}, \left\{\mathsf{M}_j[r^i_{\gamma,3}], \mathsf{K}_i[r^j_{\gamma,3}]\right\}_{j\neq i}\right) &:= \left(r^i_{\gamma,1} \oplus r^i_\beta, \left\{\mathsf{M}_1[r^i_{\gamma,1}] \oplus \mathsf{M}_1[r^i_\beta], \quad \mathsf{K}_i[r^1_{\gamma,1}] \oplus \mathsf{K}_i[r^1_\beta] \oplus \Delta_i\right\}\right)\\
&\qquad\qquad \bigcup \left\{\mathsf{M}_j[r^i_{\gamma,1}] \oplus \mathsf{M}_j[r^i_\beta], \mathsf{K}_i[r^j_{\gamma,1}] \oplus \mathsf{K}_i[r^j_\beta]\right\}_{j\neq i,1}
\end{aligned}$$

   (c) P_1 computes the following locally.

$$\begin{aligned}
\left(r^1_{\gamma,0}, \left\{\mathsf{M}_j[r^1_{\gamma,0}], \mathsf{K}_1[r^j_{\gamma,0}]\right\}_{j\neq i}\right) &:= \left(r^1_\sigma \oplus r^1_\gamma, \qquad \left\{\mathsf{M}_j[r^1_\sigma] \oplus \mathsf{M}_j[r^1_\gamma], \quad \mathsf{K}_1[r^j_\sigma] \oplus \mathsf{K}_1[r^j_\gamma]\right\}_{j\neq i}\right)\\
\left(r^1_{\gamma,1}, \left\{\mathsf{M}_j[r^1_{\gamma,1}], \mathsf{K}_1[r^j_{\gamma,1}]\right\}_{j\neq i}\right) &:= \left(r^1_{\gamma,0} \oplus r^1_\alpha, \qquad \left\{\mathsf{M}_j[r^1_{\gamma,0}] \oplus \mathsf{M}_j[r^1_\alpha], \mathsf{K}_1[r^j_{\gamma,0}] \oplus \mathsf{K}_1[r^j_\alpha]\right\}_{j\neq i}\right)\\
\left(r^1_{\gamma,2}, \left\{\mathsf{M}_j[r^1_{\gamma,2}], \mathsf{K}_1[r^j_{\gamma,2}]\right\}_{j\neq i}\right) &:= \left(r^1_{\gamma,0} \oplus r^1_\beta, \qquad \left\{\mathsf{M}_j[r^1_{\gamma,0}] \oplus \mathsf{M}_j[r^1_\beta], \mathsf{K}_1[r^j_{\gamma,0}] \oplus \mathsf{K}_1[r^j_\beta]\right\}_{j\neq i}\right)\\
\left(r^1_{\gamma,3}, \left\{\mathsf{M}_j[r^1_{\gamma,3}], \mathsf{K}_1[r^j_{\gamma,3}]\right\}_{j\neq i}\right) &:= \left(r^1_{\gamma,1} \oplus r^1_\beta \oplus 1, \left\{\mathsf{M}_j[r^1_{\gamma,1}] \oplus \mathsf{M}_j[r^1_\beta], \mathsf{K}_1[r^j_{\gamma,1}] \oplus \mathsf{K}_1[r^j_\beta]\right\}_{j\neq i}\right)
\end{aligned}$$

   (d) For each $i \neq 1$, P_i computes $\mathsf{L}^i_{\alpha,1} := \mathsf{L}^i_{\alpha,0} \oplus \Delta_i$ and $\mathsf{L}^i_{\beta,1} := \mathsf{L}^i_{\beta,0} \oplus \Delta_i$, and sends the following to P_1.

$$\begin{aligned}
G^i_{\gamma,0} &:= H\left(\mathsf{L}^i_{\alpha,0}, \mathsf{L}^i_{\beta,0}, \gamma, 0\right) \oplus \left(r^i_{\gamma,0}, \left\{\mathsf{M}_j[r^i_{\gamma,0}]\right\}_{j\neq i}, \mathsf{L}^i_{\gamma,0} \oplus \left(\bigoplus_{j\neq i} \mathsf{K}_i[r^j_{\gamma,0}]\right) \oplus r^i_{\gamma,0}\Delta_i\right)\\
G^i_{\gamma,1} &:= H\left(\mathsf{L}^i_{\alpha,0}, \mathsf{L}^i_{\beta,1}, \gamma, 1\right) \oplus \left(r^i_{\gamma,1}, \left\{\mathsf{M}_j[r^i_{\gamma,1}]\right\}_{j\neq i}, \mathsf{L}^i_{\gamma,0} \oplus \left(\bigoplus_{j\neq i} \mathsf{K}_i[r^j_{\gamma,1}]\right) \oplus r^i_{\gamma,1}\Delta_i\right)\\
G^i_{\gamma,2} &:= H\left(\mathsf{L}^i_{\alpha,1}, \mathsf{L}^i_{\beta,0}, \gamma, 2\right) \oplus \left(r^i_{\gamma,2}, \left\{\mathsf{M}_j[r^i_{\gamma,2}]\right\}_{j\neq i}, \mathsf{L}^i_{\gamma,0} \oplus \left(\bigoplus_{j\neq i} \mathsf{K}_i[r^j_{\gamma,2}]\right) \oplus r^i_{\gamma,2}\Delta_i\right)\\
G^i_{\gamma,3} &:= H\left(\mathsf{L}^i_{\alpha,1}, \mathsf{L}^i_{\beta,1}, \gamma, 3\right) \oplus \left(r^i_{\gamma,3}, \left\{\mathsf{M}_j[r^i_{\gamma,3}]\right\}_{j\neq i}, \mathsf{L}^i_{\gamma,0} \oplus \left(\bigoplus_{j\neq i} \mathsf{K}_i[r^j_{\gamma,3}]\right) \oplus r^i_{\gamma,3}\Delta_i\right)
\end{aligned}$$

Figure 3: Our main protocol instantiating $\mathcal{F}_{\mathsf{mpc}}$.

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

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 $x_w^i \oplus \lambda_w := x_w^i \left(\bigoplus_{i\in[n]} r_w^i\right)$. 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 $\left\{\left(r^i_{\gamma,\ell}, \mathsf{M}_1[r^i_{\gamma,\ell}], \mathsf{K}_1[r^i_{\gamma,\ell}]\right)\right\}_{i \neq 1}$ 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 4: Our main protocol instantiating $\mathcal{F}_{\mathsf{mpc}}$, continued.

distribution of $\{x_w^i \oplus r_w^i\}^{i \in \mathcal{H}}$ 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 $\mathbf{Hybrid_4}$ 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 $\hat{x}_w^i$ and computes $x_w^i := \hat{x}_w^i \oplus \bigoplus_{i\in[n]} r_w^i$. 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 $(r^i_w \oplus 1, \mathsf{M}_1[r^i_w] \oplus \Delta_1)$.

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 5.2, 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 5.1. *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, happens with negligible probability. $\square$

Lemma 5.2. *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. $\square$

6 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 $O(B^2 n^2)$ 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 $O(Bn^2)$.

6.1 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 potentially use inconsistent values when running $\mathcal{F}^2_{\mathsf{aBit}}$ with other parties. In our protocol shown in Figure 6, 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 6.1. *The protocol in Figure 6 securely instantiates the $\mathcal{F}^n_{\mathsf{aBit}}$ functionality with statistical security $2^{-\rho}$ in the $\mathcal{F}^2_{\mathsf{aBit}}$-hybrid model.*

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 5: Functionality $\mathcal{F}^n_{\mathsf{aBit}}$ for multi-party authenticated bit.

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

Proof. **Case 1: $P_i \in \mathcal{H}$.** Note that in this case, the only way malicious parties can break the protocol is to learn 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.

Using s_j to denote the last 2ρ bits of r in the j-th check. According to Lemma 6.1 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 send the query to $\mathcal{F}^2_{\mathsf{aBit}}$ and send the answer from $\mathcal{F}^2_{\mathsf{aBit}}$ to $\mathcal{A}$.

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 6.1. *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 $2^{-k} \times 2^l = 2^{l-k}$. ☐

6.2 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 8 for more details.

Theorem 6.2. *The protocol in Figure 8 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.

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 7: Functionality $\mathcal{F}_{\mathsf{aShare}}$ for multi-party authenticated share.

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 $K^j \leftarrow \mathsf{Open}(\mathsf{c}_{bj}, \mathsf{d}_{bj})$ 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 8: The protocol Π_{aShare} instantiating $\mathcal{F}_{\mathsf{aShare}}$.

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 6.2 □

Lemma 6.2. *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

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 9: Functionality

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

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.

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

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. □

6.3 Multi-Party Half Authenticated AND triple

Our goal is to build a leaky authenticated AND triple. The stepping stone is to first build an authenticated triple with only x's being authenticated. Later, this will be used to build a "fully" authenticated triple that is vulnerable to selective failure attacks.

Lemma 6.3. *Assuming H as a random oracle, the protocol in Figure 10 securely implements the functionality in Figure 9 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. $\square$

6.4 Multi-Party Leaky Authenticated AND Triple

Once we have multi-party version of authenticated bit, our next step is to compute leaky authenticated AND triples. The adversary can perform selective failure attacks where he gets caught with some probability.

The first step is to compute the AND triples, such that the triples will be correct if every party behaves honestly without revealing any party's share. This is done in our protocol by using $\mathcal{F}_{\mathsf{HaAND}}$. This gives the adversary a chance to perform a selective failure attack on x. The next step is to check the correctness of the computation. Note that this is main challenge. In existing protocols,

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

they follow the cut-and-choose approach using a sacrifice and merge step. Cut-and-choose ensures that only a small number of triples that are not checked are incorrect; bucketing is used twice, first to gain correctness and a second time to gain privacy, leading to the B^2n^2 term.

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 2^{-k}. This leakage can be easily eliminated using bucketing as discussed later. In the two-party protocol, one party construct "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 12.

6.4.1 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:

Protocol Π_{LaAND}

Compute the triple with semi-honest security

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 $[z^i]^i := [r^i]^i \oplus e^i$.

Check the correctness

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 it does not hold.

Figure 12: The protocol Π_{LaAND}.

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

Notice further that

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

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.

6.4.2 Unforgeability

Lemma 6.4. *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 compute, 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

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

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.

6.4.3 Proof

Theorem 6.3. *The protocol in Figure 12, 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

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

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:

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

□

6.5 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 14.

Acknowledgments

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

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

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

References

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *Advances in Cryptology—Eurocrypt 2011*, volume 6632 of *LNCS*, pages 169–188. Springer, 2011.

[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.

[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.

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

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

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

[FKOS15] Tore Kasper Frederiksen, Marcel Keller, Emmanuela Orsini, and Peter Scholl. A unified approach to MPC with preprocessing using OT. LNCS, pages 711–735. Springer, 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 *Advances in Cryptology—Eurocrypt 2017*, LNCS. Springer, 2017.

[JKO13] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In *20th ACM Conf. on Computer and Communications Security (CCS)*, pages 955–966. ACM Press, 2013.

[KOS15] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In *Advances in Cryptology—Crypto 2015, Part I*, volume 9215 of *LNCS*, pages 724–741. Springer, 2015.

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

[LOS14] Enrique Larraia, Emmanuela Orsini, and Nigel P. Smart. Dishonest majority multi-party computation for binary circuits. In *Advances in Cryptology—Crypto 2014, Part II*, volume 8617 of *LNCS*, pages 495–512. Springer, 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 *Advances in Cryptology—Crypto 2015, Part II*, volume 9216 of *LNCS*, pages 319–338. Springer, 2015.

[LSS16] Yehuda Lindell, Nigel P. Smart, and Eduardo Soria-Vazquez. More efficient constant-round multi-party computation from BMR and SHE. In *12th Theory of Cryptography Conference—TCC 2016*, LNCS, pages 554–581. Springer, 2016.

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

[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.