More Efficient MPC from Improved Triple Generation and Authenticated Garbling

Kang Yang
State Key Laboratory of Cryptology
yangk@sklc.org

Xiao Wang
Northwestern University
wangxiao@cs.northwestern.edu

Jiang Zhang
State Key Laboratory of Cryptology
jiangzhang09@gmail.com

May 7, 2020

Abstract

Recent works on distributed garbling have provided highly efficient solutions for constant-round MPC tolerating an arbitrary number of corruptions. In this work, we improve upon state-of-the-art protocols in this paradigm for further performance gain.

First, we propose a new protocol for generating authenticated AND triples, which is a key building block in many recent works.

- We propose a new authenticated bit protocol in the two-party and multi-party settings from bare IKNP OT extension, allowing us to reduce the communication by about 24% and eliminate many computation bottlenecks. We further improve the computational efficiency for multi-party authenticated AND triples with cheaper and fewer consistency checks and fewer hash function calls.
- We implemented our triple generation protocol and observe around $4\times$ to $5\times$ improvement compared to the best prior protocol in most settings. For example, in the two-party setting with 10 Gbps network and 8 threads, our protocol can generate more than 4 million authenticated triples per second, while the best prior implementation can only generate 0.8 million triples per second. In the multi-party setting, our protocol can generate more than 37000 triples per second over 80 parties, while the best prior protocol can only generate the same number of triples per second over 16 parties.

We also improve the state-of-the-art multi-party authenticated garbling protocol.

- We take the first step towards applying half-gates in the multi-party setting, which enables us to reduce the size of garbled tables by 2κ bits per gate per garbler, where κ is the computational security parameter. This optimization is also applicable in the semi-honest multi-party setting.
- We further reduce the communication of circuit authentication from 4ρ bits to 1 bit per gate, using a new multi-party batched circuit authentication, where ρ is the statistical security parameter. Prior solution with similar efficiency is only applicable in the two-party setting.

For example, in the three-party setting, our techniques can lead to roughly a 35% reduction in the size of a distributed garbled circuit.

Contents

1 Introduction

Secure multi-party computation (MPC) protocols [Yao86, GMW87] allow a set of parties with private inputs to compute a joint function without revealing anything more than the output of the function. A variety of adversarial models have been considered regarding the adversarial behaviors, the threshold of corrupt parties, etc. In this paper, we focus on statically secure MPC protocols tolerating an arbitrary number of malicious corruptions.

Distributed garbling [BMR90, DI05] allows a set of parties to jointly generate a garbled circuit in a distributed manner. It is a core tool to construct constant-round MPC protocols. Recent advances on distributed garbling have led to a set of efficient protocols [CKMZ14, LPSY15, LSS16, WRK17a, WRK17b, HSS17, HIV17, KRRW18, ZCSH18] for constant-round MPC tolerating an arbitrary number of malicious corruptions. For example, Wang et al. [WRK17b] demonstrated an implementation that can securely compute AES-128 among 32 parties in about one second, something unimaginable a few years ago. As a brief overview, these protocols all follow a similar paradigm consisting of three phases:

1. **Function-independent phase,** when parties only know an upper bound on the size of the circuit to be computed. In most protocols, this stage requires computing authenticated version of Beaver triples and takes the most computation and communication resources.

2. **Function-dependent phase,** when parties now know the function being computed. This phase usually involves generating a multi-party garbled circuit, which may be either asymmetric [WRK17b] or symmetric [HSS17].

3. **Online phase,** when parties know their inputs and can evaluate the garbled circuit generated in the previous phase.

Although this paradigm has significantly improved the efficiency of constant-round maliciously secure MPC protocols, we find that inefficiencies still exist in many key building blocks that, if optimized, can potentially lead to huge improvements.

- The **communication overhead** to obtain malicious security is still high. For example, the best-known maliciously secure two-party computation (2PC) protocol [KRRW18] still requires sending about 310 bytes per gate, even with amortization. This is about $10\times$ more communication than the best semi-honest garbled-circuit protocol [ZRE15] sending only about 32 bytes per gate.

- The **computational overhead** of existing maliciously secure protocols is surprisingly higher than commonly thought. For example, the most efficient implementation [WRK17a] in the two-party setting reports a speed of $833K$ authenticated AND triples per second under 10 Gbps network bandwidth and a 36-core CPU. However, if the network was fully used, we would expect at least $4300K$ authenticated AND triples to be generated per second,[1] which is a $5\times$ performance gap due to high computation cost!

  The problem is more prominent in the multi-party setting, where additional consistency checks are needed among the parties and this causes even more computation overhead. For example, with eight parties, the implementation from Wang et al. [WRK17b], benchmarked using the same hardware as above, can compute about $68K$ multi-party authenticated AND triples per second. The speed would be $510K$ triples per second, if the 10 Gbps network bandwidth is fully utilized. This is a performance gap of $7.5\times$.

[1] Since every authenticated AND triple in their protocol takes $\approx$ 290 bytes of communication, 10 Gbps bandwidth can support around 4.3×10^6 triples.

#Parties	2 (8 threads)	2 (32 threads)	3	8	24	40	48
[WRK17a, WRK17b]	1.26	0.59	5.02	10.77	40.55	62.66	75
This paper	0.24	0.18	1.26	2.31	6.37	11.41	13.54
Improvements	5.28×	3.28×	3.98×	4.66×	6.36×	5.5×	5.54×

Table 1: **Comparison of the best prior implementation and ours for generating an authenticated AND triple.** All reported numbers are the running time, in microseconds (μs), to generate one authenticated AND triple. All experiments are performed with machines with 10 Gbps network bandwidth and 36 vCPUs. Both implementations are applied with the same level of code optimizations.

These computation and communication overheads become increasingly prominent as the need of malicious security emerges in real life. However, as numerous prior works have extensively studied approaches to optimize this paradigm recently [WRK17a, WRK17b, HSS17, HIV17, KRRW18, ZCSH18], any improvement requires novel insights and careful analysis of the protocol.

1.1 Our Contributions

In this paper, we present a set of improvements to the distributed garbling paradigm for constant-round MPC protocols tolerating an arbitrary number of malicious corruptions. Our implementation shows significant performance boost compared to the best prior work. See Table 1 for a selected set of comparison points and Section 5 for more details. Below we summarize our results and provide intuitions of our ideas in Section 2.

First, we design a new authenticated AND triple (aAND) protocol from scratch with improved efficiency. To exploit every possible improvement, we start our analysis from the most basic building blocks, within the KOS protocol [KOS15] for maliciously secure OT extension. Our improvements can be directly applied right out of the box to many MPC protocols that need authenticated AND triples [WRK17a, WRK17b, HSS17, KRRW18, ZCSH18, AOR$^+$19, RW19, DEF$^+$19]. In detail, we improve three key components in the protocol.

- **Improved multi-party authenticated bit.** The state-of-the-art protocol [NST17] for each authenticated bit (aBit) requires $(\kappa + \rho)$-bit communication and a high computational cost due to the use of bit-matrix multiplication and multiple checks for correlation and consistency. As an evidence, two-party authenticated bit is about 3× slower than the ordinary maliciously secure OT extension even when running over a local host [Rin].

  We propose a new multi-party authenticated bit protocol directly based on the bare IKNP OT extension [IKNP03, ALSZ13] with a small *harmless* leakage. Our new protocol reduces the communication per aBit from $(\kappa + \rho)$ bits to κ bits and eliminates all sources of slowdown mentioned above. See Section 2.2 for more elaboration.

- **Improved multi-party authenticated share.** Multi-party authenticated share (aShare) is another key building block towards aAND (See Section 2.3). The previous protocol [WRK17b] for multi-party authenticated shares needs to repeat a checking procedure ρ times, to boost the soundness error from 2^{-1} to $2^{-\rho}$. We propose an improved checking procedure based on the re-randomization idea by Hazay et al. [HSS17] where a single check is sufficient.

- **Improved multi-party leaky AND triple.** Leaky authenticated AND triple (LaAND) is yet another important building block towards fully secure AND triples. We reduce the number of hash function calls by a factor of 2× when computing leaky AND triples and show that the security preserves even given the leakage introduced in aBit as above. See Section 2.4 for more details.

By applying the above optimizations, we can reduce the communication cost of authenticated AND triple generation by $\approx 24\%$. Our implementation shows that in most settings, it leads to an improvement of running time by $4\times$ to $5\times$ in both the two-party and the multi-party settings. See Table 1 for a performance summary.

Our next bundle of optimizations are specific to the WRK multi-party authenticated garbling [WRK17b] with an emphasize on the function-dependent phase.

- **Towards multi-party half-gates.** Although it is known how to distributively compute the half-gates scheme [ZRE15] in the two-party setting [KRRW18], the multi-party setting is completely open. Here we *partially* enables half-gates in the multi-party setting and reduce the size of a distributed garbled circuit from each garbler by 2κ bits per gate. Our technique here is also applicable in the semi-honest setting [BLO16]. See Section 2.5 for a high-level description.
- **Improved circuit authentication.** Katz et al. [KRRW18] proposed an efficient way to authenticate distributed garbled circuits using the amortized MAC check in the two-party setting. However, it does not directly apply to settings with more than two parties. To obtain similar efficiency in the multi-party setting, we design a new batched circuit authentication based on almost universal linear hash functions [CDD$^+$16]. The resulting solution improves the communication from 4ρ bits to 1 bit per gate. See Section 2.6 for more discussion.

As a result, for example in the three-party setting, our optimizations result in an about 35% reduction in communication for function-dependent phase.

1.2 Discussion of Some Related Works

Recently, Boyle et al. [BCG$^+$19] presented a maliciously secure protocol that can be used to realize correlated OT with sublinear communication. In the two-party setting, their protocol can be used to further reduce the communication of our protocol. Based on their performance benchmark, it would be faster than the KOS OT extension [KOS15] when the network bandwidth is below 500 Mbps. In the same security setting, Hazay, Venkitasubramaniam, and Weiss [HVW19] recently proposed a maliciously secure MPC protocol with constant communication overhead over the semi-honest GMW protocol, which implies constant-round MPC based on BMR distributed garbling, while our protocol has a communication overhead of $O\left(\rho/\log|\mathcal{C}|\right)$. However, their result remains mostly theoretical and here we are mainly interested in the practical efficiency.

1.3 Organization

In Section 2, we introduce important concepts and building blocks needed for our constructions. We also present the intuitions about how our improvements work. Then in Section 3 and Section 4, we describe in detail the improved protocol for authenticated AND triples and the improved multi-party authenticated garbling protocol, respectively. Finally, in Section 5, we discuss the concrete efficiency gain of the protocol. We defer some basic concepts and protocols/proofs to Appendix.

2 Background and Technical Overview

In this section, we introduce some background information on the state-of-the-art protocols for authenticated AND triples and authenticated garbling. Alongside, we also provide high-level ideas on how our work improves these protocols. In Appendix A, we describe more commonly known preliminaries, including two useful functionalities $\mathcal{F}_{\mathsf{Com}}$ and $\mathcal{F}_{\mathsf{Rand}}$ for commitments and coin-tossing respectively, almost universal linear hash functions, and the amortized opening procedure of authenticated bits/shares.

2.1 Notation

We use κ and ρ to denote the computational and statistical security parameters respectively. We use $a \leftarrow S$ to denote sampling a uniformly at random from a finite set S. We will use $[n]$ to denote the set $\{1, \ldots, n\}$. For a bit-string x, we use $\mathsf{lsb}(x)$ to denote the least significant bit of x, and $x[k]$ to denote the k-th bit of x. Depending on the context, we use $\{0,1\}^\kappa$, $\mathbb{F}_2^k$ and $\mathbb{F}_{2^\kappa}$ interchangeably, and thus addition in $\mathbb{F}_2^k$ and $\mathbb{F}_{2^\kappa}$ corresponds to XOR in $\{0,1\}^\kappa$. We use bold lower-case letters such as $\mathbf{x} \in \mathbb{F}_2^k$ to denote a vector, and $\mathbf{x}[k]$ to denote the k-th component of $\mathbf{x}$. We will use a hash function $\mathsf{H} : \{0,1\}^* \rightarrow \{0,1\}^\kappa$ modeled as a random oracle. We use $\mathsf{negl}(\cdot)$ to denote some unspecific negligible function such that $\mathsf{negl}(\kappa) = o(\kappa^{-c})$ for every constant c. We write $\mathbb{F}_{2^\kappa} \cong \mathbb{F}_2[X]/f(X)$ for some monic, irreducible polynomial $f(X)$ of degree κ. We denote by $P_1, \ldots, P_n$ the parties.

A *boolean circuit* $\mathcal{C}$ is represented as a list of gates of the form $(\alpha, \beta, \gamma, T)$, which denotes a gate with input-wire indices α and β, output-wire index γ and gate type $T \in \{\oplus, \wedge\}$. By $|\mathcal{C}|$, we denote the number of AND gates in a circuit $\mathcal{C}$. We use $\mathcal{I}_i$ to denote the set of *circuit-input wire* indices with the input from party P_i, $\mathcal{W}$ to denote the set of output wire indices for all AND gates, and $\mathcal{O}_i$ to denote the set of *circuit-output wire* indices associated with the output of P_i. Without loss of generality, we assume that the input and output of all parties have the same length, i.e., $|\mathcal{I}_1| = \cdots = |\mathcal{I}_n|$ and $|\mathcal{O}_1| = \cdots = |\mathcal{O}_n|$. We use $|\mathcal{I}|$ and $|\mathcal{O}|$ to denote the length of *all* circuit-input wires and circuit-output wires respectively. Our MPC protocol will use $\mathsf{H}(\{Z_w\}_{w \in \mathcal{W}})$ to denote $\mathsf{H}(Z_{w_1}, \ldots, Z_{w_{|\mathcal{C}|}})$ for $Z_w \in \{0,1\}^\kappa$, where here $w_1, \ldots, w_{|\mathcal{C}|}$ are the sorted output wire indices of all AND gates in an increasing order.

In this paper, we consider a static, malicious adversary who can corrupt up to $n-1$ out of n parties. We use $A \subset [n]$ to denote the set of all corrupt parties. All our protocols allow abort, and are provably secure in the standard simulation-based security model [Gol04]. Our protocols need a broadcast channel, which can be efficiently implemented using a standard 2-round echo-broadcast protocol [GL05] as we allow abort. The communication of this broadcast protocol can be optimized in a batch [DPSZ12] by using either an almost universal linear hash function or a collision-resistant hash function.

2.2 Multi-Party Authenticated Bits

Authenticated bits (or equivalently information-theoretic MACs) were firstly proposed for maliciously secure two-party computation by Nielsen et al. [NNOB12], and can also be extended to the multi-party setting [BDOZ11, LOS14]. Every party P_i holds a uniform *global key* $\Delta_i \in \{0,1\}^\kappa$. We say that a party P_i holds a bit $x \in \{0,1\}$ authenticated by P_j, if P_j holds a random *local key* $\mathsf{K}_j[x] \in \{0,1\}^\kappa$ and P_i holds the MAC $\mathsf{M}_j[x] = \mathsf{K}_j[x] \oplus x\Delta_j$. We write $[x]_i^j = (x, \mathsf{M}_j[x], \mathsf{K}_j[x])$ to represent a two-party authenticated bit where x is known to P_i and authenticated to only one party P_j. In the multi-party setting, we let $[x]_i = (x, \{\mathsf{M}_j[x]\}_{j \neq i}, \{\mathsf{K}_j[x]\}_{j \neq i})$ denote a multi-party authenticated bit, where the bit x is known by P_i and authenticated to all other parties. In more detail, P_i holds $(x, \{\mathsf{M}_j[x]\}_{j \neq i})$, and P_j holds $\mathsf{K}_j[x]$ for $j \neq i$.

We note that $[x]_i$ is XOR-homomorphic. That is, for two authenticated bits $[x]_i$ and $[y]_i$ held by P_i, it is possible to locally compute an authenticated bit $[z]_i$ with $z = x \oplus y$ by each party locally XOR their respective values. That is, 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 computes $\mathsf{K}_j[z] := \mathsf{K}_j[x] \oplus \mathsf{K}_j[y]$ for each $j \neq i$. We use $[z]_i := [x]_i \oplus [y]_i$ to denote the above operation. As such, $[x]_i^j$ is also XOR-homomorphic.

With a slight abuse of the notation, we can also authenticate a constant bit b: P_i sets $\{\mathsf{M}_j[b] := 0\}_{j \neq i}$; P_j sets $\mathsf{K}_j[b] := b\Delta_j$ for each $j \neq i$. Similarly, let $[b]_i = (b, \{\mathsf{M}_j[b]\}_{j \neq i}, \{\mathsf{K}_j[b]\}_{j \neq i})$. Now we can write $[x]_i \oplus b = [x]_i \oplus [b]_i$, and let $b[x]_i$ be equal to $[0]_i$ if $b = 0$ and $[x]_i$ otherwise.

Prior solution. Prior solution for multi-party authenticated bits is very complicated, involving the following steps to generate a bit known by P_i and authenticated to all other parties:

1. First, using the maliciously secure KOS OT extension [KOS15], P_i computes random correlated strings with P_j, e.g., $\mathsf{M}_j[x]$ and $\mathsf{K}_j[x]$ such that $\mathsf{M}_j[x] \oplus \mathsf{K}_j[x] = x\Delta_j$. This includes an IKNP OT extension [IKNP03, ALSZ13] followed by a KOS correlation check [KOS15] for the consistency of the choice-bit vector. The correlation check is leaky in which it allows the adversary to guess a few bits of P_j's global key Δ_j.

2. To establish two-party authenticated bits, Nielsen et al. [NST17] proposed a way to eliminate the above leakage. We can execute the first step to obtain random correlated strings of length $(\kappa + \rho)$ bits. Then, we can use a random bit matrix to compress the bit string to κ bits and at the same time eliminate the leakage.

3. To generate multi-party authenticated bits for P_i, we can execute the above two steps between P_i and each other party, where the KOS correlation check and bit-matrix multiplication compression are executed $n-1$ times for each authenticated bit.

4. The above procedure for multi-party authenticated bits is not fully secure, as a malicious P_i may use inconsistent bits when executing the two-party authenticated bit protocol with different parties. Wang et al. [WRK17a] designed an extra consistency check that allows honest parties to catch such inconsistent behavior with probability $1/2$. The check needs to be repeated by ρ times to ensure a cheating probability of $1/2^\rho$.

In practice, the above steps are very costly in computation. Different layers of consistency checks and compression cause heavy computation and require the data to flow through the CPU cache back and forth. The computation of bit-matrix multiplication is particularly expensive, even after carefully optimized. For example, libOTe [Rin] shows that two-party authenticated bit is about $3\times$ slower than the ordinary maliciously secure OT extension even when running on the same machine.

Our solution. Towards improving the efficiency of the above protocol, we make the following crucial observations:

1. The leakage caused by the KOS correlation check is *harmless* in our setting because, intuitively, the resulting correlated strings will be used either for authentication or for constructing distributed garbled circuits, where learning all bits of a global key is required to break the security. In particular, an adversary can guess a few bits of honest parties' global keys but get caught if any guess is incorrect. Therefore, the probability that the protocol does not abort and the adversary learns the whole global key is bounded by $2^{-c} \times 2^{-(\kappa-c)} = 2^{-\kappa}$ for the leakage of c bits, which is the same as the case without such leakage.

2. The two consistency checks are of similar goals and thus one may already achieve the goal of the other. In detail, both checks aim to ensure that a malicious receiver uses consistent choice bits: the first KOS consistency check is to ensure that a unique choice-bit vector is used among all columns of the extension matrix within *one execution between two parties* such that the unique choice-bit vector can be extracted by the simulator; while the second multi-party check is to ensure that a consistent choice-bit vector is used across *two or more executions between multiple parties*.

As a result, we propose an improved multi-party authenticated bit protocol that allows the adversary to guess a small number of bits of the global keys of honest parties with the risk of being caught if any guess is incorrect. The protocol consists of only two steps: 1) P_i executes the IKNP OT extension protocol [ALSZ13] with every party P_j for $j \neq i$; 2) All parties jointly execute *a single check* that serves both purposes of correlation check and consistency check in the prior work. The check works similarly with the KOS correlation check, except that it is done jointly by all parties in a batch.

Compared with prior solutions, we improve the communication overhead and computation cost per authenticated bit as follows: 1) reduce the communication from $\kappa + \rho$ bits to κ bits; 2) reduce the number of base OTs between each pair of parties from $\kappa + \rho$ to κ; 3) eliminate the need of bit-matrix multiplication, as well as the multi-party consistency check. The detailed protocol and proof of security can be found in Section 3.1.

2.3 Multi-Party Authenticated Shares

In most cases, authenticating a bit known to one party is not sufficient. We would like a way to authenticate a bit unknown to all parties, which can be done by secret sharing together with authenticating each share. In detail, to generate an authenticated secret bit x, we can generate XOR shares of x (i.e., shares $\{x^i\}_{i=1}^n$ such that $\bigoplus_{i=1}^n x^i = x$), and then ask every party to authenticate to every other parties about their shares. We use $\langle x \rangle = \big([x^1]_1, \ldots [x^n]_n\big)$ to denote an authenticated share of bit x, i.e., $\langle x \rangle$ means that every party P_i holds $\big(x^i, \{\mathsf{M}_j[x^i], \mathsf{K}_i[x^j]\}_{j \neq i}\big)$. It is straightforward to see that authenticated shares are also XOR-homomorphic. For a constant bit $b \in \{0,1\}$, we let $\langle x \rangle \oplus b = \big([x^1]_1 \oplus b, [x^2]_2, \ldots, [x^n]_n\big)$, and define $b\langle x \rangle$ to be equal to $\langle 0 \rangle = ([0]_1, \ldots, [0]_n)$ if $b = 0$ and $\langle x \rangle$ otherwise.

Prior solution. In prior works, authenticated shares are constructed by letting each party execute the authenticated bit protocol with their only shares of the secret bit. However, since every party participates in multiple authentication processes, it is possible that a malicious party uses different global keys in multiple executions of the authenticated bit protocol with different parties, and thus causes the inconsistency. In WRK [WRK17b], they proposed a protocol to check the consistency of global keys by making use of the XOR-homomorphic property of authenticated bits. Their checking protocol requires each party to compute $2\rho + 1$ commitments.

Our solution. Based on the re-randomization technique by Hazay et al. [HSS17], we improve the WRK consistency check for authenticated shares by reducing the number of commitments from $2\rho + 1$ to 1. In particular, we use a linear map that maps κ random shares of a party P_i to a random element $\mathbf{y}^i$ in $\mathbb{F}_{2^\kappa}$. Then we use a random zero-share to re-randomize each element $\mathbf{y}^i$. To prevent the collusion, each party needs to make only *a single* commitment. Note that the inconsistency may occur only when there are at least two honest parties. In this case, $\mathbf{y}^i$ is kept secret from the re-randomization based on zero-share, which assures the consistency of global keys.

The checking procedure can also be efficiently implemented given hardware support for finite field multiplications. See Section 3.2 for the detailed protocol and proof of security.

2.4 Improved Authenticated AND triples

The protocol for leaky authenticated AND triples is to generate a random authenticated AND triple $\big(\langle x \rangle, \langle y \rangle, \langle z \rangle\big)$ with one caveat that the adversary can choose to guess the share x^i of an honest party P_i. A correct guess remains undetected, while an incorrect guess will be caught.

Prior solution. The multi-party leaky AND triple protocol by Wang et al. [WRK17b] consists of two steps: 1) the parties execute a protocol to generate AND triples without correctness guarantee; 2) all parties run a checking procedure to ensure correctness, which also introduce some potential leakage to the adversary. Recently, Katz et al. [KRRW18] proposed an efficient checking protocol reducing the number of hash function calls by half in the two-party setting. The key idea is to apply the point-and-permute technique [BMR90] for garbled circuits to the context of AND triple generation. They integrated the above two steps into one as the least significant bit can represent the underlying share.

Our solution. We extend their idea from two-party setting to the multi-party setting. The extension of the protocol is fairly straightforward; nevertheless, we believe it is an important task to figure out all details of the security proof. We give the protocol description and a full proof that the protocol is still provably secure given the leakage of global keys introduced as above in Appendix D.1.

2.5 Improved Distributed Garbling with Partial Half-Gates

Classical and half-gates garbling. The classical garbling with point-and-permute [BMR90] and free-XOR [KS08] requires 4 garbled rows per AND gate. Let P_2 and P_1 be the garbler and evaluator respectively. Each wire w is associated with a random garbled label $\mathsf{L}_{w,0} \in \{0,1\}^\kappa$ and a wire mask $\lambda_w \in \{0,1\}$ both known only to the garbler. The garbled label for a bit b is defined as $\mathsf{L}_{\alpha,b} = \mathsf{L}_{\alpha,0} \oplus b\Delta_2$, where Δ_2 is a random global offset only known to P_2. The garbled table computed by P_2, namely $\{G_{uv}\}_{u,v\in\{0,1\}}$, for an AND gate $(\alpha, \beta, \gamma, \wedge)$ consists of four garbled rows in the following form

$$G_{uv} := \mathsf{H}(\mathsf{L}_{\alpha,u}, \mathsf{L}_{\beta,v}) \oplus \mathsf{L}_{\gamma,0} \oplus r_{uv}\Delta_2,$$

where $r_{uv} = (u \oplus \lambda_\alpha) \wedge (v \oplus \lambda_\beta) \oplus \lambda_\gamma$, and we omit γ in H for simplicity. Half-gates by Zahur et al. [ZRE15] is the state-of-the-art garbling scheme that only requires 2 garbled rows per AND gate. In this case, the garbled table can be written as:

$$\begin{aligned} G_0 :=& \mathsf{H}(\mathsf{L}_{\alpha,0}) \oplus \mathsf{H}(\mathsf{L}_{\alpha,1}) \oplus \lambda_\beta\Delta_2, \\ G_1 :=& \mathsf{H}(\mathsf{L}_{\beta,0}) \oplus \mathsf{H}(\mathsf{L}_{\beta,1}) \oplus \mathsf{L}_{\alpha,0} \oplus \lambda_\alpha\Delta_2. \end{aligned}$$

Garbler P_2 can compute the 0-label for the output wire as:

$$\mathsf{L}_{\gamma,0} := \mathsf{H}(\mathsf{L}_{\alpha,0}) \oplus \mathsf{H}(\mathsf{L}_{\beta,0}) \oplus (\lambda_\alpha\lambda_\beta \oplus \lambda_\gamma)\Delta_2.$$

Classical and half-gates two-party distributed garbling. Following the observation by Katz et al. [KRRW18], we can conceptually divide the authenticated garbling protocol into two parts: 1) jointly generate a distributed garbled circuit among all parties; 2) authenticate the correctness of the garbled circuit for the evaluator. Here we only consider the first part about distributed garbling. The WRK distributed garbling [WRK17a] in the two-party setting can be written as:

$$\begin{aligned} P_2 : G^2_{uv} &:= \mathsf{H}(\mathsf{L}_{\alpha,u}, \mathsf{L}_{\beta,v}) \oplus \big(\mathsf{L}_{\gamma,0} \oplus \mathsf{K}_2[r^1_{uv}] \oplus r^2_{uv}\Delta_2\big) \\ P_1 : G^1_{uv} &:= \mathsf{M}_2[r^1_{uv}], \end{aligned}$$

where $r^1_{uv} \oplus r^2_{uv} = r_{uv}$ is defined as above. The correctness can be checked given the fact that $G^1_{uv} \oplus G^2_{uv} = G_{uv}$ as in the classical garbling.

Recently, Katz et al. [KRRW18] showed that the half-gates technique can be applied to the above two-party distributed garbling. Although P_2 cannot compute G_0, G_1 and $\mathsf{L}_{\gamma,0}$ as in the half-gates garbling (because P_2 does not know the wire masks, and thus cannot compute the terms $\lambda_\beta\Delta_2$, $\lambda_\alpha\Delta_2$ and $(\lambda_\alpha\lambda_\beta \oplus \lambda_\gamma)\Delta_2$), both parties P_1 and P_2 hold the authenticated shares, say, $R_1 \oplus R_2 = \lambda_\beta\Delta_2$, $S_1 \oplus S_2 = \lambda_\alpha\Delta_2$, and $T_1 \oplus T_2 = (\lambda_\alpha\lambda_\beta \oplus \lambda_\gamma)\Delta_2$. Thus, they can conceptually "shift" the entire garbling procedure by R_1, S_1 and T_1. In detail, P_2 can compute

$$\begin{aligned} G^2_0 &:= \mathsf{H}(\mathsf{L}_{\alpha,0}) \oplus \mathsf{H}(\mathsf{L}_{\alpha,1}) \oplus R_2, \\ G^2_1 &:= \mathsf{H}(\mathsf{L}_{\beta,0}) \oplus \mathsf{H}(\mathsf{L}_{\beta,1}) \oplus \mathsf{L}_{\alpha,0} \oplus S_2, \\ \mathsf{L}_{\gamma,0} &:= \mathsf{H}(\mathsf{L}_{\alpha,0}) \oplus \mathsf{H}(\mathsf{L}_{\beta,0}) \oplus T_2. \end{aligned}$$

Evaluator P_1 can recover G_0 and G_1 by computing $G_0 := G^2_0 \oplus R_1$ and $G_1 := G^2_1 \oplus S_1$. Then P_1 can perform the standard half-gates evaluation, and adds T_1 as a correction value, so as to compute the garbled label for output wire γ.

Applying half-gates for multi-party authenticated garbling. Applying half-gates to the multi-party setting has been an open problem proposed by multiple prior works [BLO16, WRK17b, BLO17, KRRW18, BJPR18]. We present how to partially use half-gates in the multi-party distributed garbling.

Let's first recall the classical multi-party distributed garbling [WRK17b]. For each wire w, every garbler P_i ($i \geq 2$) has a pair of garbled labels $\mathsf{L}^i_{w,0}, \mathsf{L}^i_{w,1}$ such that $\mathsf{L}^i_{w,0} \oplus \mathsf{L}^i_{w,1} = \Delta_i$, where Δ_i is a random offset only known to P_i. For each AND gate $(\alpha, \beta, \gamma, \wedge)$ and $u, v \in \{0,1\}$, the distributed garbling is constructed in the following form:

$$\begin{aligned} P_i, i \geq 2 : G^i_{uv} &:= \mathsf{H}(\mathsf{L}^i_{\alpha,u}, \mathsf{L}^i_{\beta,v}) \oplus \Big(\{\mathsf{M}_j[r^i_{uv}]\}_{j\neq i,1}, \mathsf{L}^i_{\gamma,0} \oplus \big(\textstyle\bigoplus_{j\neq i}\mathsf{K}_i[r^j_{uv}]\big) \oplus r^i_{uv}\Delta_i\Big), \\ P_1 : G^1_{uv} &:= \{\mathsf{M}_j[r^1_{uv}]\}_{j\neq 1}, \end{aligned}$$

where $\bigoplus_{i\in[n]} r^i_{uv} = r_{uv}$ is defined as above.

As we can see above, the multi-party garbling is very complicated and difficult to analyze. Our first step is to further split the distributed garbled table into two parts as below:

$$\begin{aligned} A^i_{uv} &:= \mathsf{H}(\mathsf{L}^i_{\alpha,u}, \mathsf{L}^i_{\beta,v}) \oplus \big(\mathsf{L}^i_{\gamma,0} \oplus (\textstyle\bigoplus_{j\neq i}\mathsf{K}_i[r^j_{uv}]) \oplus r^i_{uv}\Delta_i\big), \\ B^i_{uv} &:= \mathsf{H}'(\mathsf{L}^i_{\alpha,u}, \mathsf{L}^i_{\beta,v}) \oplus \big(\{\mathsf{M}_j[r^i_{uv}]\}_{j\neq i,1}\big). \end{aligned}$$

Essentially, we can view G^i_{uv} as (A^i_{uv}, B^i_{uv}). Now we can see that A^i_{uv} is very similar to the two-party distributed garbling. Thus we can attempt to apply the half-gates optimization on this portion:

$$\begin{aligned} A^i_0 &:= \mathsf{H}(\mathsf{L}^i_{\alpha,0}) \oplus \mathsf{H}(\mathsf{L}^i_{\alpha,1}) \oplus R_i, \\ A^i_1 &:= \mathsf{H}(\mathsf{L}^i_{\beta,0}) \oplus \mathsf{H}(\mathsf{L}^i_{\beta,1}) \oplus \mathsf{L}^i_{\alpha,0} \oplus S_i, \\ \mathsf{L}^i_{\gamma,0} &:= \mathsf{H}(\mathsf{L}^i_{\alpha,0}) \oplus \mathsf{H}(\mathsf{L}^i_{\beta,0}) \oplus T_i, \end{aligned}$$

where $\bigoplus_{i\in[n]} R_i = \lambda_\beta\Delta_i$, $\bigoplus_{i\in[n]} S_i = \lambda_\alpha\Delta_i$ and $\bigoplus_{i\in[n]} T_i = (\lambda_\alpha\lambda_\beta \oplus \lambda_\gamma)\Delta_i$. Unlike the two-party setting, here P_1 cannot recover $\mathsf{H}(\mathsf{L}^i_{\alpha,0}) \oplus \mathsf{H}(\mathsf{L}^i_{\alpha,1}) \oplus \lambda_\beta\Delta_i$ and $\mathsf{H}(\mathsf{L}^i_{\beta,0}) \oplus \mathsf{H}(\mathsf{L}^i_{\beta,1}) \oplus \mathsf{L}^i_{\alpha,0} \oplus \lambda_\alpha\Delta_i$, and then perform the standard half-gates evaluation, since it does not get the other parties' shares for $\lambda_\beta\Delta_i$ and $\lambda_\alpha\Delta_i$. By a careful evaluation, we show that evaluator P_1 can still compute the garbled label for output wire γ in the following way. If P_1 holds public values $\Lambda_\alpha, \Lambda_\beta$ and the corresponding garbled labels $\{\mathsf{L}^i_{\alpha,\Lambda_\alpha}, \mathsf{L}^i_{\beta,\Lambda_\beta}\}_{i\neq 1}$, then for each $i \neq 1$, it computes as follows:

1. Evaluate the half-gates portion:

$$\begin{aligned} &\mathsf{H}(\mathsf{L}^i_{\alpha,\Lambda_\alpha}) \oplus \mathsf{H}(\mathsf{L}^i_{\beta,\Lambda_\beta}) \oplus \Lambda_\alpha \cdot A^i_0 \oplus \Lambda_\beta \cdot (A^i_1 \oplus \mathsf{L}^i_{\alpha,\Lambda_\alpha}) \\ &= \mathsf{H}(\mathsf{L}^i_{\alpha,0}) \oplus \mathsf{H}(\mathsf{L}^i_{\beta,0}) \oplus \Lambda_\alpha R_i \oplus \Lambda_\beta S_i \oplus \Lambda_\alpha\Lambda_\beta\Delta_i. \end{aligned}$$

2. Evaluate classical garbling portion. Let $u = \Lambda_\alpha$ and $v = \Lambda_\beta$. Then, the evaluator P_1 can compute

$$\big\{\mathsf{M}_j[r^i_{uv}]\big\}_{j\neq i,1} := \mathsf{H}'(\mathsf{L}^i_{\alpha,\Lambda_\alpha}, \mathsf{L}^i_{\beta,\Lambda_\beta}) \oplus B^i_{uv},$$

   where $\mathsf{M}_j[r^i_{uv}]$ is P_i's share of $\Lambda_\gamma\Delta_j$ for $j \neq i$.

3. P_1 can compute its share $\mathsf{M}_i[r^1_{uv}]$ of $\Lambda_\gamma\Delta_i$ for each $i \neq 1$. Then, P_1 combines them with the above results as follows:

$$\begin{aligned} &\big(\mathsf{H}(\mathsf{L}^i_{\alpha,0}) \oplus \mathsf{H}(\mathsf{L}^i_{\beta,0}) \oplus \Lambda_\alpha R_i \oplus \Lambda_\beta S_i \oplus \Lambda_\alpha\Lambda_\beta\Delta_i\big) \oplus \big(\textstyle\bigoplus_{j\neq i}\mathsf{M}_i[r^j_{uv}]\big) \\ &= \mathsf{H}(\mathsf{L}^i_{\alpha,0}) \oplus \mathsf{H}(\mathsf{L}^i_{\beta,0}) \oplus T_i \oplus \Lambda_\gamma\Delta_i = \mathsf{L}^i_{\gamma,0} \oplus \Lambda_\gamma\Delta_i = \mathsf{L}^i_{\gamma,\Lambda_\gamma}. \end{aligned}$$

The correctness holds because

$$\begin{aligned}\Lambda_\gamma\Delta_i &= \Lambda_\alpha\Lambda_\beta\Delta_i \oplus \Lambda_\alpha\lambda_\beta\Delta_i \oplus \Lambda_\beta\lambda_\alpha\Delta_i \oplus (\lambda_\alpha\lambda_\beta \oplus \lambda_\gamma)\Delta_i \\ &= \Lambda_\alpha\Lambda_\beta\Delta_i \oplus \Lambda_\alpha\big(\textstyle\bigoplus_{i\in[n]} R_i\big) \oplus \Lambda_\beta\big(\textstyle\bigoplus_{i\in[n]} S_i\big) \oplus \big(\textstyle\bigoplus_{i\in[n]} T_i\big) \\ &= \big(\Lambda_\alpha\Lambda_\beta\Delta_i \oplus \Lambda_\alpha R_i \oplus \Lambda_\beta S_i \oplus T_i\big) \oplus \big(\textstyle\bigoplus_{j\neq i}(\Lambda_\alpha R_j \oplus \Lambda_\beta S_j \oplus T_j)\big) \\ &= \big(\Lambda_\alpha\Lambda_\beta\Delta_i \oplus \Lambda_\alpha R_i \oplus \Lambda_\beta S_i \oplus T_i\big) \oplus \big(\textstyle\bigoplus_{j\neq i} \mathsf{M}_i[r^j_{uv}]\big),\end{aligned}$$

where $\Lambda_\gamma = (\Lambda_\alpha \oplus \lambda_\alpha) \wedge (\Lambda_\beta \oplus \lambda_\beta) \oplus \lambda_\gamma$ and $\mathsf{M}_i[r^j_{uv}] = \Lambda_\alpha R_j \oplus \Lambda_\beta S_j \oplus T_j$ is P_j's share of $\Lambda_\gamma\Delta_i$. As a result, we can reduce the communication per AND gate from each garbler by 2κ bits in the function-dependent phase. We refer the reader to Section 4 for the detailed construction.

2.6 Batch Circuit Authentication in the Multi-Party Setting

In this section, we focus on the *circuit authentication* part, which is used to authenticate the correctness of a garbled circuit. Specifically, this part roughly works as follows:

- In the preprocessing phase, for each AND gate $(\alpha, \beta, \gamma, \wedge)$, every party P_i holds authenticated shares of λ_α, λ_β, λ_γ and $\lambda_{\alpha\beta} = \lambda_\alpha \cdot \lambda_\beta$.
- After evaluating the distributed garbled circuit in the online phase, for each wire w, the evaluator P_1 obtains a *public value* Λ_w, which is the XOR of the actual value on the wire (based on the input) and a wire mask λ_w. P_1 would like to check correctness of all public values by using the above authenticated shares. In particular, it will guarantee that for each AND gate, the actual values on the wires form an AND relationship.

Prior solution. For each AND gate $(\alpha, \beta, \gamma, \wedge)$ and $u, v \in \{0, 1\}$, we define $r_{uv} = (u \oplus \lambda_\alpha) \wedge (v \oplus \lambda_\beta) \oplus \lambda_\gamma$. In the original WRK protocol [WRK17b], the circuit authentication was essentially done by encrypting authenticated bits of the form $(r^i_{uv}, \mathsf{M}_1[r^i_{uv}])$ in each garbled row, where r^i_{uv} is P_i's share of r_{uv}. This because the garblers do not know the public values at the stage of garbling. When incorporating the optimization [WRK17a] into the protocol, their solution requires 4ρ bits of communication per AND gate in the function-dependent phase.

Katz et al. [KRRW18] observed that in the two-party setting, such circuit authentication can be done in a batch, which reduces the communication to 1 bit per AND gate. In particular, evaluator P_1 needs to send the public values on the output wires of all AND gates to P_2, as P_2 cannot evaluate the circuit. For each AND gate $(\alpha, \beta, \gamma, \wedge)$, for correctness of Λ_γ, it suffices to show that $t_\gamma = (\Lambda_\alpha \oplus \lambda_\alpha) \wedge (\Lambda_\beta \oplus \lambda_\beta) \oplus (\Lambda_\gamma \oplus \lambda_\gamma) = 0$. Two parties compute the authenticated shares t^1_γ and t^2_γ of t_γ by using the authenticated shares of λ_α, λ_β, λ_γ and $\lambda_\alpha \cdot \lambda_\beta$. Then P_2 sends $\mathsf{M}_1[t^2_\gamma]$ to P_1, who checks its validity by comparing it with $\mathsf{K}_1[t^2_\gamma] \oplus t^1_\gamma\Delta_1$, where $t_\gamma = 0$ if and only if $t^1_\gamma = t^2_\gamma$. This authentication procedure can be made in a batch for all AND gates by checking whether $\mathsf{H}(\{\mathsf{M}_1[t^2_w]\}_{w\in\mathcal{W}}) = \mathsf{H}(\{\mathsf{K}_1[t^2_w] \oplus t^1_w\Delta_1\}_{w\in\mathcal{W}})$. A malicious P_1 may flip some public values, and reveals some secret shares held by P_2 from such authentication, which may break the privacy. To prevent the attack, P_1 also needs to send $\mathsf{H}(\{\mathsf{M}_2[t^1_w]\}_{w\in\mathcal{W}})$ to P_2 who checks that it is equal to $\mathsf{H}(\{\mathsf{K}_2[t^1_w] \oplus t^2_w\Delta_2\}_{w\in\mathcal{W}})$. This solution does not extend to the multi-party setting directly, because when there are multiple garblers, P_1 only knows t^1_γ but not individual t^i_γ for $i \neq 1$.

Our solution. For each wire $w \in \mathcal{W}$, we let P_1 check that $t_w = \bigoplus_{i\in[n]} t^i_w = 0$, after P_1 sends $\{\Lambda_w\}_{w\in\mathcal{W}}$ to all other parties, where t_w is defined as above. In a naive approach, each garbler P_i sends $(t^i_w, \mathsf{M}_1[t^i_w])$ to P_1, who checks that $\mathsf{M}_1[t^i_w] = \mathsf{K}_1[t^i_w] \oplus t^i_w\Delta_1$. This requires $|\mathcal{C}| \cdot (\kappa + 1)$ bits of communication per garbler. By optimizing the approach with batched MAC check, the communication is reduced to $|\mathcal{C}| + \kappa$ bits.

We propose a new circuit authentication procedure in the multi-party setting based on an almost-universal linear hash function $\mathbf{H}$ (as defined in Appendix A.2), which further reduces the communication per garbler to κ bits. Specifically, each garbler P_i $(i \neq 1)$ sends $z_i = \mathbf{H}(\{\mathsf{M}_1[t_w^i]\}_{w \in \mathcal{W}}) \in \mathbb{F}_{2^\kappa}$ to P_1, and P_1 computes $\mathsf{M}_1[t_w^1] := \bigoplus_{i \neq 1} \mathsf{K}_1[t_w^i] \oplus t_w^1 \Delta_1$ for each wire $w \in \mathcal{W}$. For each $w \in \mathcal{W}$, $t_w = 0$ if and only if

$$\bigoplus_{i \in [n]} \mathsf{M}_1[t_w^i] = t_w^1 \Delta_1 \oplus \bigoplus_{i \neq 1} \left(\mathsf{K}_1[t_w^i] \oplus \mathsf{M}_1[t_w^i]\right) = \bigoplus_{i \in [n]} t_w^i \Delta_1 = t_w \Delta_1 = 0.$$

P_1 computes $z_1 := \mathbf{H}(\{\mathsf{M}_1[t_w^1]\}_{w \in \mathcal{W}})$, and then checks that $\bigoplus_{i \in [n]} z_i = 0$. As $\mathbf{H}$ is XOR-homomorphic, we have that

$$\bigoplus_{i \in [n]} z_i = \bigoplus_{i \in [n]} \mathbf{H}(\{\mathsf{M}_1[t_w^i]\}_{w \in \mathcal{W}}) = \mathbf{H}(\{\bigoplus_{i \in [n]} \mathsf{M}_1[t_w^i]\}_{w \in \mathcal{W}}) = 0.$$

From the almost-universal property, we have $\bigoplus_{i \in [n]} \mathsf{M}_1[t_w^i] = 0$ for $w \in \mathcal{W}$. We can use a polynomial hash based on GMAC to instantiate the linear hash function $\mathbf{H}$, whose computation is blazing fast given hardware-instruction support.

To prevent the attack mentioned above, we let P_1 send $h_i = \mathsf{H}(\{\mathsf{L}^i_{w,\Lambda_w}\}_{w \in \mathcal{W}})$ to every garbler P_i who checks that $h_i = \mathsf{H}(\{\mathsf{L}^i_{w,0} \oplus \Lambda_w \Delta_i\}_{w \in \mathcal{W}})$. Through the approach, every garbler P_i can check the correctness of the public values sent by P_1, because evaluator P_1 can learn only one garbled label for each wire and garbled label $\mathsf{L}^i_{w,\Lambda_w}$ can be viewed as an MAC on bit Λ_w. After the circuit authentication procedure, all parties can obtain the correct public values, which allows our protocol to support multi-output functions in a straightforward way. We give the detailed protocol in Section 4 and the full proof in Appendix E.

2.7 Other Optimization

In the WRK protocol [WRK17b], the input bits are masked with authenticated shares. However we observe that this is not necessary and that an extended form of authenticated bits is already sufficient. Intuitively, since every party can arbitrarily choose its input, and thus the shares from all other parties can be set to 0. We define a useful operation called $\mathsf{Bit2Share}$, which takes as input an authenticated bit $[\lambda]_i = \left(\lambda, \{\mathsf{M}_k[\lambda]\}_{k \neq i}, \{\mathsf{K}_k[\lambda]\}_{k \neq i}\right)$ with bit λ known by P_i, and extends it to an authenticated share $\langle \lambda \rangle$ as follows:

- Set $[\lambda^i]_i := [\lambda]_i$: P_i sets $\lambda^i := \lambda$ and $\left\{\mathsf{M}_k[\lambda^i] := \mathsf{M}_k[\lambda]\right\}_{k \neq i}$, P_k sets $\mathsf{K}_k[\lambda^i] := \mathsf{K}_k[\lambda]$ for each $k \neq i$;
- Set $[\lambda^j]_j := [0]_j$ for each $j \neq i$: P_j sets $\lambda^j := 0$ and $\{\mathsf{M}_k[\lambda^j] := 0\}_{k \neq j}$, and P_k defines $\mathsf{K}_k[\lambda^j] := 0$ for each $k \neq j$.

In our protocol, the circuit-input wires are processed using the above procedure instead of a full-fledged authenticated shares. This is partially effective when the input is large (See Section 5). A similar idea is used in the semi-honest MPC protocol [BLO16], where the MACs need *not* to be considered.

3 Improved Preprocessing Protocols

In this section, we present the details of our optimizations for faster authenticated AND triple generation. Since we have discussed the key insights and high-level ideas of the preprocessing protocols in Section 2, here we will focus on detailed description of the protocols and their proofs of security.

In Section 3.1, we will present our improved multi-party authenticated bit protocol with a detailed proof of security. Then in Section 3.2, we will show the authenticated share protocol with an improved global-key consistency check. We defer the detailed description and security proof of the improved protocol for authenticated AND triples to Appendix D.

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

This functionality runs with parties $P_1, \ldots, P_n$. Let $A \subset [n]$ be the set of corrupt parties.

Initialize: Upon receiving (init) from all parties, sample $\Delta_i \leftarrow \{0,1\}^\kappa$ for $i \notin A$ and receive $\Delta_i \in \{0,1\}^\kappa$ from the adversary for $i \in A$. Store Δ_i for $i \in [n]$ and send Δ_i to party P_i.

Macro $\mathsf{AuthBit}(i, x)$ (this is an internal subroutine only)

1. If P_i is corrupted, receive an MAC $\mathsf{M}_j[x] \in \{0,1\}^\kappa$ from the adversary and compute $\mathsf{K}_j[x] := \mathsf{M}_j[x] \oplus x\Delta_j$ for each $j \neq i$.
2. Otherwise, sample honest parties' keys $\mathsf{K}_j[x] \leftarrow \{0,1\}^\kappa$ for $j \notin A, j \neq i$. Receive keys $\mathsf{K}_j[x]$ for $j \in A$ from the adversary. Compute the MACs $\mathsf{M}_j[x] := \mathsf{K}_j[x] \oplus x\Delta_j$ for $j \neq i$.
3. Output $(x, \{\mathsf{M}_j[x]\}_{j \neq i})$ to P_i and $\mathsf{K}_j[x]$ to P_j for $j \neq i$.

Authenticated bits/shares/triples: This functionality generates random authenticated bits, shares and triples as follows:

1. Upon receiving (aBit, i) from all parties, sample $x \leftarrow \{0,1\}$ if P_i is honest and receive $x \in \{0,1\}$ otherwise, and then generate a (random) authenticated bit $[x]_i$ by executing $\mathsf{AuthBit}(i, x)$.
2. Upon receiving (aShare) from all parties, sample $x \leftarrow \{0,1\}$, and then execute the following macro $\mathsf{AuthShare}(x)$ to generate a random authenticated share $\langle x \rangle$:
   - Receive $x^i \in \{0,1\}$ from the adversary for $i \in A$; sample $x^i \leftarrow \{0,1\}$ for $i \notin A$ such that $\bigoplus_{i \in [n]} x^i = x$.
   - For each $i \in [n]$, execute $\mathsf{AuthBit}(i, x^i)$.
3. Upon receiving (aAND) from all parties, sample random bits $a, b \leftarrow \{0,1\}$, compute $c := a \wedge b$, and generate a random authenticated AND triple $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ via running $\mathsf{AuthShare}(x)$ for each $x \in \{a, b, c\}$.

Selective failure leakage: Wait for the adversary to input $(\mathsf{leak}, i, S, \{\Delta'[k]\}_{k \in S})$. If P_i is honest, this functionality executes the macro $\mathsf{GKleak}(i, S, \{\Delta'[k]\}_{k \in S})$ defined as follows:

- If there exists some $k \in S$ such that $\Delta'[k] \neq \Delta_i[k]$, this functionality sends `fail` to all parties and aborts.
- Otherwise, it sends `success` to the adversary and proceeds as if nothing has happened.

Figure 1: The multi-party preprocessing functionality.

Our protocols for authenticated bits, shares and AND triples jointly implement the preprocessing functionality $\mathcal{F}_{\mathsf{prep}}$ as shown in Figure 1. In functionality $\mathcal{F}_{\mathsf{prep}}$, we allow the adversary to make multiple (leak) queries on the same index $i \notin A$. Each bit guess of Δ_i made by the adversary will be caught with probability $1/2$. For each (leak) query, the adversary needs to provide a subset $S \subseteq [\kappa]$ representing the positions in which the guessed bits locate. Our MPC protocol with improved authenticated garbling as shown in Section 4 will work in the $\mathcal{F}_{\mathsf{prep}}$-hybrid model.

3.1 Optimized Multi-Party Authenticated Bits

We propose a new protocol Π_{aBit} to generate authenticated bits in the multi-party malicious setting. Our protocol uses a *correlated OT with errors* functionality $\mathcal{F}_{\mathsf{COTe}}$ [KOS15] shown in Figure 2. In functionality $\mathcal{F}_{\mathsf{COTe}}$, if a receiver P_R is honest, it will input a "monochrome" vector $\mathbf{x}_i = x_i \cdot (1, \ldots, 1)$ for $i \in [\ell]$ and $x_i \in \{0,1\}$, which results in the correct correlation, i.e., $\mathsf{M}[x_i] = \mathsf{K}[x_i] + x_i \cdot \Delta$. If P_R is malicious, it may input a "polychromatic" vector $\mathbf{x}_i \neq x_i \cdot (1, \ldots, 1)$ for $i \in [\ell]$, which results in $\mathsf{M}[x_i] = \mathsf{K}[x_i] + \mathbf{x}_i * \Delta$, where $\mathbf{x}_i * \Delta = (\mathbf{x}_i[1] \cdot \Delta[1], \ldots, \mathbf{x}_i[\kappa] \cdot \Delta[\kappa])$. We can rewrite $\mathbf{x}_i = x_i \cdot (1, \ldots, 1) + \mathbf{e}_i$, and get $\mathsf{M}[x_i] =$

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

Initialize: Upon receiving (init, Δ) from a sender P_S where global key $\Delta \in \mathbb{F}_2^\kappa$, and (init) from a receiver P_R, store Δ and ignore all subsequent (init) commands.

Extend: Upon receiving $(\mathsf{extend}, \ell, \mathbf{x}_1, \ldots, \mathbf{x}_\ell)$ from P_R where $\mathbf{x}_i \in \mathbb{F}_2^\kappa$, and (extend, ℓ) from P_S, this functionality does the following:

1. For each $i \in [\ell]$, sample $\mathsf{K}[x_i] \leftarrow \mathbb{F}_2^\kappa$. If P_S is corrupted, instead receive $\mathsf{K}[x_i] \in \mathbb{F}_2^\kappa$ from the adversary.
2. Compute $\mathsf{M}[x_i] := \mathsf{K}[x_i] + \mathbf{x}_i * \Delta \in \mathbb{F}_2^\kappa$ for each $i \in [\ell]$, where $*$ denotes component-wise product.
3. If P_R is corrupted, receive $\mathsf{M}[x_i] \in \mathbb{F}_2^\kappa$ from the adversary and recompute $\mathsf{K}[x_i] := \mathsf{M}[x_i] + \mathbf{x}_i * \Delta$.
4. For each $i \in [\ell]$, output $\mathsf{K}[x_i]$ to P_S and $\mathsf{M}[x_i]$ to P_R.

Figure 2: Functionality for correlated OT with errors.

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

This functionality generates random bits known by a party P_i and authenticated to all other parties.

Initialize: Upon receiving (init, i) from P_i and $(\mathsf{init}, i, \Delta_j)$ from P_j for $j \neq i$ where $\Delta_j \in \{0,1\}^\kappa$, store Δ_j for $j \neq i$ and ignores all subsequent (init) commands.

Authentication: Upon receiving (aBit, i, ℓ) from all parties, sample $x_1, \ldots, x_\ell \leftarrow \{0,1\}$ if P_i is honest, and receive bits $\{x_k\}_{k\in[\ell]}$ from the adversary otherwise. Generate ℓ authenticated bits $\{[x_k]_i\}_{k\in[\ell]}$ by running $\mathsf{AuthBit}(i, x_k)$ defined in Figure 1 for $k \in [\ell]$.

Selective failure leakage: If P_i is corrupted, wait for the adversary to input $(\mathsf{leak}, j, S, \{\Delta'[k]\}_{k\in S})$. If P_j is honest, this functionality executes $\mathsf{GKleak}(j, S, \{\Delta'[k]\}_{k\in S})$ as defined in Figure 1.

Figure 3: Functionality for multi-party authenticated bits.

$\mathsf{K}[x_i]+x_i\cdot\Delta+\mathbf{e}_i*\Delta$, where $\mathbf{e}_i \in \mathbb{F}_2^\kappa$ is an error vector counting the number of positions in which P_R cheated. An efficient protocol, which implements the functionality $\mathcal{F}_{\mathsf{COTe}}$, has been described in [Nie07, KOS15]. This protocol is the same as the IKNP OT extension protocol [IKNP03, ALSZ13], except that it terminates before hashing the output with the random oracle to break the correlation and executing the final round of communication. Nielsen [Nie07] has shown that the protocol securely realizes the functionality $\mathcal{F}_{\mathsf{COTe}}$.

We present the details of our protocol Π_{aBit} in Figure 4, where Π_{aBit} works in the $(\mathcal{F}_{\mathsf{COTe}}, \mathcal{F}_{\mathsf{Rand}})$-hybrid model. The bits sampled by a party P_i in our protocol are authenticated by *weak* global keys, where the adversary is allowed to guess a few bits on the global keys of honest parties. We use a functionality $\mathcal{F}_{\mathsf{aBit}}$ shown in Figure 3 to define authenticated bits with selective failure leakage of global keys. In the (init) command of $\mathcal{F}_{\mathsf{aBit}}$, an honest party P_j will input a random global key Δ_j, while a corrupt party P_j allows to send an arbitrary string $\Delta_j \in \{0,1\}^\kappa$. By the (leak) command of $\mathcal{F}_{\mathsf{aBit}}$, the adversary may guess a few bits of global key Δ_j for $j \notin \mathcal{A}$. A correct guess keeps undetected, while an incorrect guess will be caught. In particular, while the adversary succeeds to leak c_j bits of Δ_j with probability 2^{-c_j} for some $c_j \in [\kappa] \cup \{0\}$, the remaining $\kappa - c_j$ bits of Δ_j are still uniform and unknown from the adversary's view.

Security of our authenticated bit protocol. We first analyze the checking procedure (Steps 5–7) of protocol Π_{aBit} shown in Figure 4 and give several important lemmas. The analysis and lemmas can be found in Appendix B.1. Then, we prove the security of our authenticated bit protocol Π_{aBit} in the following theorem. The full formal proof of the theorem is postponed to Appendix B.2.

Protocol Π_{aBit}

Let $\ell' = \ell + (\kappa + \rho)$. A party P_i generates ℓ bits authenticated by all other parties.

Initialize: All parties initialize the protocol as follows:

1. For each $j \neq i$, P_j picks a random global key $\Delta_j \leftarrow \{0,1\}^\kappa$.
2. For $j \neq i$, P_j sends $(\mathsf{init}, \Delta_j)$ to $\mathcal{F}_{\mathsf{COTe}}$; P_i sends (init) to $\mathcal{F}_{\mathsf{COTe}}$.

Generate authenticated bits: The parties generate ℓ' authenticated bits without correctness guarantee as follows:

3. P_i picks random bits $x_1, \ldots, x_{\ell'} \leftarrow \{0,1\}$, and then sets a monochrome vector $\mathbf{x}_k := x_k \cdot (1, \ldots, 1) \in \mathbb{F}_2^\kappa$ for each $k \in [\ell']$.
4. For each $j \neq i$, P_j and P_i call $\mathcal{F}_{\mathsf{COTe}}$ on respective inputs (extend, ℓ') and $(\mathsf{extend}, \ell', \mathbf{x}_1, \ldots, \mathbf{x}_{\ell'})$, which returns the keys $\{\mathsf{K}_j[x_k]\}_{k \in [\ell']}$ to P_j and the MACs $\{\mathsf{M}_j[x_k]\}_{k \in [\ell']}$ to P_i.

Check correlation and consistency: The parties check the correlation of their outputs from $\mathcal{F}_{\mathsf{COTe}}$ and the consistency of P_i's inputs in multiple calls of $\mathcal{F}_{\mathsf{COTe}}$ as follows:

5. The parties call coin-tossing functionality $\mathcal{F}_{\mathsf{Rand}}$ to obtain ℓ' random coefficients $\chi_1, \ldots, \chi_{\ell'} \in \mathbb{F}_{2^\kappa}$.
6. P_i locally computes over $\mathbb{F}_{2^\kappa}$ the value $\mathbf{y}^i := \sum_{k=1}^{\ell'} \chi_k \cdot x_k$, and $\left\{\mathsf{M}_j[\mathbf{y}^i] := \sum_{k=1}^{\ell'} \chi_k \cdot \mathsf{M}_j[x_k]\right\}_{j \neq i}$ and broadcasts $\mathbf{y}^i$ to all parties. For each $j \neq i$, P_i also sends $\mathsf{M}_j[\mathbf{y}^i]$ to P_j.
7. For each $j \neq i$, P_j computes $\mathsf{K}_j[\mathbf{y}^i] := \sum_{k=1}^{\ell'} \chi_k \cdot \mathsf{K}_j[x_k]$, and checks that $\mathsf{M}_j[\mathbf{y}^i] = \mathsf{K}_j[\mathbf{y}^i] + \mathbf{y}^i \cdot \Delta_j$. If the check fails, P_j aborts.
8. The parties output ℓ authenticated bits $[x_1]_i, \ldots, [x_\ell]_i$ with $[x_k]_i = (x_k, \mathsf{M}_j[x_k], \mathsf{K}_j[x_k])$ for $k \in [\ell]$.

Figure 4: Protocol for generating multi-party aBits.

Theorem 1. *Protocol Π_{aBit} shown in Figure 4 securely realizes functionality $\mathcal{F}_{\mathsf{aBit}}$ in the $(\mathcal{F}_{\mathsf{COTe}}, \mathcal{F}_{\mathsf{Rand}})$-hybrid model.*

Optimization and communication complexity. We first optimize the generation of random coefficients in Step 5 of protocol Π_{aBit} described in Figure 4. Specifically, instead of calling functionality $\mathcal{F}_{\mathsf{Rand}}$, the parties can use the Fiat-Shamir heuristic to compute the random coefficients $\{\chi_k\}_{k \in [\ell']}$ by hashing the transcript during the authenticated bit generation procedure, which is secure in the random oracle model. This optimization reduces the communication rounds of protocol Π_{aBit} by two rounds.

Now, we analyze the rounds and communication cost of protocol Π_{aBit} with the above optimization, including the rounds and cost of the IKNP OT extension protocol [ALSZ13] realizing $\mathcal{F}_{\mathsf{COTe}}$. When ignoring the communication rounds in the initialization phase, our protocol Π_{aBit} needs only one round. If we adopt the base OT protocols such as [PVW08, CO15, CSW20] to implement the initialization procedure, the whole protocol has three rounds. We ignore the communication cost of global keys setup in the initialization phase (Step 1 and Step 2), as the setup needs to be run only once, and the one-time setup cost depends on which base OT protocol is used and is *minor* for the efficiency of the whole protocol. The communication cost of protocol Π_{aBit} is about $(n-1)(\ell + \kappa + \rho)\kappa$ bits.

3.2 Improved Multi-Party Authenticated Shares

We propose an efficient protocol Π_{aShare}, which allows n parties to compute authenticated shares of secret bits. One straightforward approach is to call $\mathcal{F}_{\mathsf{aBit}}$ n times, where in the j-th call, the parties obtain a random authenticated bit $[x^j]_j$ for a random bit x^j known only by P_j. However, a malicious party P_i may

Protocol Π_{aShare}

Initialize: All parties initialize the protocol as follows:

1. For each $i \in [n]$, P_i picks a random global key $\Delta_i \leftarrow \{0,1\}^\kappa$.
2. For each $i \in [n]$, for each $j \neq i$, P_j sends $(\mathsf{init}, i, \Delta_j)$ to $\mathcal{F}_{\mathsf{aBit}}$.

Generate authenticated shares: All parties generate $\ell + \kappa$ authenticated shares without consistency guarantee by calling $\mathcal{F}_{\mathsf{aBit}}$.

3. For each $i \in [n]$, all parties send $(\mathsf{aBit}, i, \ell + \kappa)$ to $\mathcal{F}_{\mathsf{aBit}}$, which samples $x^i_1, \ldots, x^i_\ell, r^i_1, \ldots, r^i_\kappa \leftarrow \{0,1\}$ and sends random authenticated bits $\{[x^i_k]_i\}_{k\in[\ell]}$ and $\{[r^i_h]_i\}_{h\in[\kappa]}$ to the parties.

   If receiving `fail` from functionality $\mathcal{F}_{\mathsf{aBit}}$, the parties abort.

Consistency check: The parties check the consistency of global keys.

4. Each party P_i locally computes over $\mathbb{F}_{2^\kappa}$ the following values:

$$\mathbf{y}^i := \sum_{h=1}^{\kappa} r^i_h \cdot X^{h-1},\ \mathsf{M}_j[\mathbf{y}^i] := \sum_{h=1}^{\kappa} X^{h-1} \cdot \mathsf{M}_j[r^i_h] \text{ for } j \neq i,\ \mathsf{K}_i[\mathbf{y}^j] := \sum_{h=1}^{\kappa} X^{h-1} \cdot \mathsf{K}_i[r^j_h] \text{ for } j \neq i.$$

5. Every party P_i obtains a random zero-share $\mathbf{u}^i \in \mathbb{F}_{2^\kappa}$ such that $\sum_{i=1}^n \mathbf{u}^i = 0$ by exchanging random elements over a private channel as follows:
   - For each $i \in [n]$ and $j \neq i$, P_i picks a random element $\mathbf{u}^{i,j} \leftarrow \mathbb{F}_{2^\kappa}$ and privately sends it to P_j.
   - Every party P_i computes $\mathbf{u}^i := \sum_{j\neq i} \left(\mathbf{u}^{i,j} + \mathbf{u}^{j,i}\right)$ over $\mathbb{F}_{2^\kappa}$.
6. Every party P_i computes $\tilde{\mathbf{y}}^i := \mathbf{y}^i + \mathbf{u}^i$, and then broadcasts it to all parties. Then, for each $i \in [n]$, P_i computes $\mathbf{y} := \sum_{i=1}^n \tilde{\mathbf{y}}^i$.
7. Each party P_i computes $\mathbf{z}^i_i := \sum_{j\neq i} \mathsf{K}_i[\mathbf{y}^j] + (\mathbf{y}^i + \mathbf{y}) \cdot \Delta_i$, and commits to $\left(\{\mathbf{z}^i_j := \mathsf{M}_j[\mathbf{y}^i]\}_{j\neq i}, \mathbf{z}^i_i\right) \in \mathbb{F}^n_{2^\kappa}$ by calling the (Commit) command of $\mathcal{F}_{\mathsf{Com}}$.
8. After all commitments have been made, all parties open their commitments by calling the (Open) command of $\mathcal{F}_{\mathsf{Com}}$, and then check that:

$$\text{for each } i \in [n],\ \sum_{j=1}^{n} \mathbf{z}^j_i = 0.$$

   If the check fails, the parties abort.

9. The parties output ℓ authenticated shares $\langle x_1 \rangle, \ldots, \langle x_\ell \rangle$ with $\langle x_k \rangle = ([x^1_k]_1, \ldots, [x^n_k]_n)$ for $k \in [\ell]$.

Figure 5: Protocol for generating multi-party authenticated shares.

use inconsistent global keys in multiple calls of $\mathcal{F}_{\mathsf{aBit}}$. This results in that two authenticated bits $[x^{j_0}]_{j_0}$ and $[x^{j_1}]_{j_1}$ are authenticated by two different global keys Δ_i and Δ'_i respectively. Based on a similar observation [WRK17b], we note that the two-party functionality $\mathcal{F}_{\mathsf{COTe}}$ has already guaranteed that P_i uses the same global key Δ_i, when P_i and P_j generate the MACs on multiple bits. Therefore, if one authenticated share has the consistent global keys, then all authenticated shares have the consistent global keys. In our construction, we let all parties additionally generate κ authenticated shares, and then open them to check the consistency of global keys.

The detailed construction of authenticated share protocol Π_{aShare} is described in Figure 5. If $r^i_1, \ldots, r^i_\kappa \in$

$\{0,1\}$ are sampled at random, then $\mathbf{y}^i = \sum_{h=1}^{\kappa} r_h^i \cdot X^{h-1}$ is uniformly random over $\mathbb{F}_{2^\kappa}$. In the following theorem, we prove that protocol Π_{aShare} securely realizes the functionality $\mathcal{F}_{\mathsf{aShare}}$ for authenticated shares with weak global keys as shown in Figure 12 of Appendix C. The full proof of the theorem is provided in Appendix C. We also give the communication rounds and complexity of protocol Π_{aShare} in Appendix C.

Theorem 2. *Protocol Π_{aShare} shown in Figure 5 securely realizes functionality $\mathcal{F}_{\mathsf{aShare}}$ in the $(\mathcal{F}_{\mathsf{aBit}}, \mathcal{F}_{\mathsf{Com}})$-hybrid model.*

Implementing the (aBit) command of $\mathcal{F}_{\mathsf{prep}}$. In functionality $\mathcal{F}_{\mathsf{prep}}$ shown in Figure 1, all parties can call the (aBit) command to generate authenticated bits in which the bits are known by n different parties. The parties can execute the protocol Π_{aBit} described in Figure 4 n times to implement this command. In this case, a malicious party P_i may use inconsistent global keys in n different executions of Π_{aBit}. Nevertheless, we note that if one authenticated share has the consistent global keys, then all multi-party authenticated bits have also the consistent global keys. Therefore, by one execution of protocol Π_{aShare}, we have already guaranteed that all authenticated bits from n executions of Π_{aBit} have the consistent global keys. In a very special case that Π_{aShare} is not executed by the parties when the circuit does not include any AND gate, the parties still need to perform the consistency check underlying the protocol Π_{aShare} to guarantee the consistency of global keys.

4 Optimized Multi-Party Authenticated Garbling

4.1 Construction in the $\mathcal{F}_{\mathsf{prep}}$-hybrid model and Proof of Security

In this section, we present our MPC protocol Π_{mpc} in the $\mathcal{F}_{\mathsf{prep}}$-hybrid model. Since we have already discussed the main ideas of our improvements in Section 2, we directly show the complete description of the protocol in Figure 6 and Figure 7. In protocol Π_{mpc}, we use an amortized opening process for authenticated bits/shares described in Appendix A.3, which has also been used in the previous protocols such as [NNOB12, KRRW18]. Specifically, every party can send the bits/shares along with a hash value of the corresponding MACs to the other parties, which implements the amortized opening procedure denoted by Open. In Appendix A.3, we prove that the amortized opening is still secure in our setting where a few bits of global keys may be leaked via the selective failure attack.

In addition, protocol Π_{mpc} uses an almost universal linear hash function $\mathbf{H}(\mathbf{z}) = \mathbf{z}[1]\cdot\chi + \cdots + \mathbf{z}[m]\cdot\chi^m$, where $\mathbf{z} \in \mathbb{F}_{2^\kappa}^m$ and $m = |\mathcal{C}|$. The definition of such hash functions and the security of the construction are given in Appendix A.2. In the output processing of protocol Π_{mpc}, without loss of generality, we assume that every party's output is associated with different circuit-output wires. If we allow two parties to obtain the same output from the same circuit-output wires such as $\mathcal{O}_i = \mathcal{O}_j$, then for each $w \in \mathcal{O}_i$ the wire masks λ_w^i and λ_w^j need to be revealed over a private channel.

In Appendix E, we give a detailed security proof of protocol Π_{mpc}. In particular, we are able to prove the following result.

Theorem 3. *Let $f : \{0,1\}^{|\mathcal{I}|} \to \{0,1\}^{|\mathcal{O}|}$ be an n-party functionality. Then protocol Π_{mpc} shown in Figures 6 and 7 securely computes f in the presence of a static malicious adversary corrupting up to $n-1$ parties in the $\mathcal{F}_{\mathsf{prep}}$-hybrid model, where H is a random oracle.*

4.2 Communication Complexity

In this section, we first give the communication complexity of our protocol Π_{mpc} shown in Figures 6 and 7, and then compare it with the state-of-the-art constant-round maliciously secure MPC protocols [HSS17, WRK17b] in the dishonest majority setting.

Protocol Π_{mpc}

Inputs: In the function-independent phase, all parties know $|\mathcal{C}|$ and $|\mathcal{I}|$. In the function-dependent phase, the parties agree on a circuit $\mathcal{C}$ for a function $f : \{0,1\}^{\mathcal{I}_1} \times \cdots \times \{0,1\}^{\mathcal{I}_n} \to \{0,1\}^{\mathcal{O}_1} \times \cdots \times \{0,1\}^{\mathcal{O}_n}$. In the online phase, P_i holds an input $x^i \in \{0,1\}^{\mathcal{I}_i}$ for every $i \in [n]$, where x^i_w denotes the bit of input x^i on a circuit-input wire $w \in \mathcal{I}_i$.

Function-independent phase:

1. All parties send (init) to $\mathcal{F}_{\mathsf{prep}}$, which returns a random $\Delta_i \in \{0,1\}^\kappa$ to P_i for each $i \in [n]$ with $\mathsf{lsb}(\Delta_2) = 1$.
2. For each $i \in [n]$ and $w \in \mathcal{I}_i$, the parties send (aBit, i) to $\mathcal{F}_{\mathsf{prep}}$, which returns a random authenticated bit $[\lambda_w]_i$ to the parties. Then the parties define an authenticated share $\langle \lambda_w \rangle$ via running $\mathsf{Bit2Share}([\lambda_w]_i)$.
3. For each $w \in \mathcal{W}$, the parties send (aShare) to $\mathcal{F}_{\mathsf{prep}}$, which returns a random authenticated share $\langle \lambda_w \rangle$ to them.
4. For each $w \in \mathcal{W}$, the parties send (aAND) to $\mathcal{F}_{\mathsf{prep}}$, which returns a random authenticated AND triple $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ to the parties.
5. For each $w \in \mathcal{I}_1 \cup \cdots \cup \mathcal{I}_n$, P_i samples $\mathsf{L}^i_{w,0} \leftarrow \{0,1\}^\kappa$ for $i \neq 1$.

   If receiving `fail` from $\mathcal{F}_{\mathsf{prep}}$ in Steps 2–4, the parties abort.

Function-dependent phase:

6. For each XOR gate $(\alpha, \beta, \gamma, \oplus)$, the parties compute $\langle \lambda_\gamma \rangle := \langle \lambda_\alpha \rangle \oplus \langle \lambda_\beta \rangle$. For $i \neq 1$, P_i also computes $\mathsf{L}^i_{\gamma,0} := \mathsf{L}^i_{\alpha,0} \oplus \mathsf{L}^i_{\beta,0}$.
7. For all AND gates $(\alpha, \beta, \gamma, \wedge)$, the parties execute in parallel:
   (a) Take a fresh authenticated AND triple $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ from the previous phase, and then compute $\langle d \rangle := \langle \lambda_\alpha \rangle \oplus \langle a \rangle$ and $\langle e \rangle := \langle \lambda_\beta \rangle \oplus \langle b \rangle$.
   (b) Compute $d := \mathsf{Open}(\langle d \rangle)$ and $e := \mathsf{Open}(\langle e \rangle)$.
   (c) Compute $\langle \lambda_{\alpha\beta} \rangle = \langle \lambda_\alpha \cdot \lambda_\beta \rangle := \langle c \rangle \oplus d \cdot \langle b \rangle \oplus e \cdot \langle a \rangle \oplus d \cdot e$.
8. For each AND gate $(\alpha, \beta, \gamma, \wedge)$, for $i \neq 1$, 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 computes the following:

$$\mathcal{G}^i_{\gamma,0} := \mathsf{H}(\mathsf{L}^i_{\alpha,0}, \gamma) \oplus \mathsf{H}(\mathsf{L}^i_{\alpha,1}, \gamma) \oplus (\textstyle\bigoplus_{j \neq i} \mathsf{K}_i[\lambda^j_\beta]) \oplus \lambda^i_\beta \Delta_i$$

$$\mathcal{G}^i_{\gamma,1} := \mathsf{H}(\mathsf{L}^i_{\beta,0}, \gamma) \oplus \mathsf{H}(\mathsf{L}^i_{\beta,1}, \gamma) \oplus \mathsf{L}^i_{\alpha,0} \oplus (\textstyle\bigoplus_{j \neq i} \mathsf{K}_i[\lambda^j_\alpha]) \oplus \lambda^i_\alpha \Delta_i$$

$$\mathsf{L}^i_{\gamma,0} := \mathsf{H}(\mathsf{L}^i_{\alpha,0}, \gamma) \oplus \mathsf{H}(\mathsf{L}^i_{\beta,0}, \gamma) \oplus (\textstyle\bigoplus_{j \neq i} \mathsf{K}_i[\lambda^j_{\alpha\beta}]) \oplus \lambda^i_{\alpha\beta} \Delta_i \oplus (\textstyle\bigoplus_{j \neq i} \mathsf{K}_i[\lambda^j_\gamma]) \oplus \lambda^i_\gamma \Delta_i$$

$b_\gamma := \mathsf{lsb}(\mathsf{L}^i_{\gamma,0})$ if $i = 2$. For $u, v \in \{0,1\}$, compute the following:

$$\left\{\mathsf{M}_j[r^i_{uv}] := u \cdot \mathsf{M}_j[\lambda^i_\beta] \oplus v \cdot \mathsf{M}_j[\lambda^i_\alpha] \oplus \mathsf{M}_j[\lambda^i_{\alpha\beta}] \oplus \mathsf{M}_j[\lambda^i_\gamma]\right\}_{j \neq i,1}$$

$$G^{i,j}_{\gamma,00} := \mathsf{H}(\mathsf{L}^i_{\alpha,0}, \mathsf{L}^i_{\beta,0}, \gamma, j) \oplus \mathsf{M}_j[r^i_{00}] \text{ for } j \neq i, 1$$

$$G^{i,j}_{\gamma,01} := \mathsf{H}(\mathsf{L}^i_{\alpha,0}, \mathsf{L}^i_{\beta,1}, \gamma, j) \oplus \mathsf{M}_j[r^i_{01}] \text{ for } j \neq i, 1$$

$$G^{i,j}_{\gamma,10} := \mathsf{H}(\mathsf{L}^i_{\alpha,1}, \mathsf{L}^i_{\beta,0}, \gamma, j) \oplus \mathsf{M}_j[r^i_{10}] \text{ for } j \neq i, 1$$

$$G^{i,j}_{\gamma,11} := \mathsf{H}(\mathsf{L}^i_{\alpha,1}, \mathsf{L}^i_{\beta,1}, \gamma, j) \oplus \mathsf{M}_j[r^i_{11}] \text{ for } j \neq i, 1$$

   For each wire $w \in \mathcal{W}$, every garbler P_i sends $\left(\mathcal{G}^i_{w,0}, \mathcal{G}^i_{w,1}, \left\{G^{i,j}_{w,00}, G^{i,j}_{w,01}, G^{i,j}_{w,10}, G^{i,j}_{w,11}\right\}_{j \neq i,1}\right)$ to P_1. Additionally P_2 sends $\{b_w\}_{w \in \mathcal{W}}$ to P_1.

Figure 6: Our MPC protocol in the $\mathcal{F}_{\mathsf{prep}}$-hybrid model.

Protocol Π_{mpc}, continued

Online phase:

9. For each $i \in [n]$ and $w \in \mathcal{I}_i$, the parties execute as follows:
   (a) P_i computes $\Lambda_w := x_w^i \oplus \lambda_w$, and then broadcasts Λ_w to all parties.
   (b) For each $j \neq 1$, P_j computes and sends $\mathsf{L}_{w,\Lambda_w}^j := \mathsf{L}_{w,0}^j \oplus \Lambda_w \Delta_j$ to P_1.
10. P_1 evaluates the circuit following the topological order. For each gate $(\alpha, \beta, \gamma, T)$, P_1 holds $(\Lambda_\alpha, \{\mathsf{L}_{\alpha,\Lambda_\alpha}^i\}_{i\neq 1})$ and $(\Lambda_\beta, \{\mathsf{L}_{\beta,\Lambda_\beta}^i\}_{i\neq 1})$, and
   - If $T = \oplus$, compute $\Lambda_\gamma := \Lambda_\alpha \oplus \Lambda_\beta$ and $\mathsf{L}_{\gamma,\Lambda_\gamma}^i := \mathsf{L}_{\alpha,\Lambda_\alpha}^i \oplus \mathsf{L}_{\beta,\Lambda_\beta}^i$ for $i \neq 1$.
   - If $T = \wedge$, let $u = \Lambda_\alpha$ and $v = \Lambda_\beta$, and compute the following:
     (a) For each $j \neq 1$, $\mathsf{M}_j[r_{uv}^1] := \Lambda_\alpha \cdot \mathsf{M}_j[\lambda_\beta^1] \oplus \Lambda_\beta \cdot \mathsf{M}_j[\lambda_\alpha^1] \oplus \mathsf{M}_j[\lambda_{\alpha\beta}^1] \oplus \mathsf{M}_j[\lambda_\gamma^1]$.
     (b) For $i \neq 1$ and $j \neq i, 1$, $\mathsf{M}_j[r_{uv}^i] := \mathsf{H}(\mathsf{L}_{\alpha,\Lambda_\alpha}^i, \mathsf{L}_{\beta,\Lambda_\beta}^i, \gamma, j) \oplus G_{\gamma,uv}^{i,j}$.
     (c) For each $i \neq 1$, compute the garbled label on the output wire:
     $$\mathsf{L}_{\gamma,\Lambda_\gamma}^i := \mathsf{H}(\mathsf{L}_{\alpha,\Lambda_\alpha}^i, \gamma) \oplus \mathsf{H}(\mathsf{L}_{\beta,\Lambda_\beta}^i, \gamma) \oplus \Lambda_\alpha \mathcal{G}_{\gamma,0}^i \oplus \Lambda_\beta(\mathcal{G}_{\gamma,1}^i \oplus \mathsf{L}_{\alpha,\Lambda_\alpha}^i) \oplus \left(\bigoplus_{j\neq i} \mathsf{M}_i[r_{uv}^j]\right).$$
     (d) Compute the public value $\Lambda_\gamma := b_\gamma \oplus \mathsf{lsb}(\mathsf{L}_{\gamma,\Lambda_\gamma}^2)$.
11. P_1 computes $h_i := \mathsf{H}(\{\mathsf{L}_{w,\Lambda_w}^i\}_{w\in\mathcal{W}})$ for each $i \neq 1$, and also samples a seed $\chi \leftarrow \mathbb{F}_{2^\kappa}$. For $i \neq 1$, P_1 sends $\left(\{\Lambda_w\}_{w\in\mathcal{W}}, h_i, \chi\right)$ to P_i. Then, P_i checks that $h_i = \mathsf{H}(\{\mathsf{L}_{w,0}^i \oplus \Lambda_w \Delta_i\}_{w\in\mathcal{W}})$. If the check fails, P_i aborts. For each XOR gate $(\alpha, \beta, \gamma, \oplus)$ and $i \neq 1$, P_i computes locally $\Lambda_\gamma := \Lambda_\alpha \oplus \Lambda_\beta$.
12. For all AND gates $(\alpha, \beta, \gamma, \wedge)$, P_1 checks $t_\gamma = (\Lambda_\alpha \oplus \lambda_\alpha) \wedge (\Lambda_\beta \oplus \lambda_\beta) \oplus (\Lambda_\gamma \oplus \lambda_\gamma) = 0$ in a batch, by interacting with all other parties as follows:
   (a) For each AND gate $(\alpha, \beta, \gamma, \wedge)$ and $i \neq 1$, P_i computes
   $$\mathsf{M}_1[t_\gamma^i] := \Lambda_\alpha \cdot \mathsf{M}_1[\lambda_\beta^i] \oplus \Lambda_\beta \cdot \mathsf{M}_1[\lambda_\alpha^i] \oplus \mathsf{M}_1[\lambda_{\alpha\beta}^i] \oplus \mathsf{M}_1[\lambda_\gamma^i].$$
   (b) For each AND gate $(\alpha, \beta, \gamma, \wedge)$, P_1 computes $t_\gamma^1 := \Lambda_\alpha \cdot \Lambda_\beta \oplus \Lambda_\gamma \oplus \Lambda_\alpha \cdot \lambda_\beta^1 \oplus \Lambda_\beta \cdot \lambda_\alpha^1 \oplus \lambda_{\alpha\beta}^1 \oplus \lambda_\gamma^1$ and the following values:
   $$\left\{\mathsf{K}_1[t_\gamma^i] := \Lambda_\alpha \cdot \mathsf{K}_1[\lambda_\beta^i] \oplus \Lambda_\beta \cdot \mathsf{K}_1[\lambda_\alpha^i] \oplus \mathsf{K}_1[\lambda_{\alpha\beta}^i] \oplus \mathsf{K}_1[\lambda_\gamma^i]\right\}_{i\neq 1}, \ \mathsf{M}_1[t_\gamma^1] := \left(\bigoplus_{i\neq 1} \mathsf{K}_1[t_\gamma^i]\right) \oplus t_\gamma^1 \Delta_1.$$
   (c) Let $\mathbf{H}$ be the almost universal linear hash function defined by χ. For each $i \neq 1$, P_i computes and sends $z_i := \mathbf{H}(\{\mathsf{M}_1[t_w^i]\}_{w\in\mathcal{W}}) \in \mathbb{F}_{2^\kappa}$ to P_1.
   (d) P_1 computes $z_1 := \mathbf{H}(\{\mathsf{M}_1[t_w^1]\}_{w\in\mathcal{W}}) \in \mathbb{F}_{2^\kappa}$, and checks that $\sum_{i=1}^n z_i = 0$. If the check fails, P_1 aborts.
13. For each $i \in [n]$, P_i computes its output as follows:
   (a) For each wire $w \in \mathcal{O}_i$ and $j \neq i$, P_j and P_i compute $\lambda_w^j := \mathsf{Open}([\lambda_w^j]_j^i)$.
   (b) For each wire $w \in \mathcal{O}_i$, P_i computes $y_w^i := \Lambda_w \oplus \left(\bigoplus_{j\in[n]} \lambda_w^j\right)$. Then P_i outputs y^i.

Figure 7: Our MPC protocol in the $\mathcal{F}_{\mathsf{prep}}$-hybrid model, continued.

In the function-independent phase, our protocol Π_{mpc} needs 8 communication rounds, when the initialization procedure for the setup of global keys is instantiated by the base OT protocols such as [PVW08,

Func. Ind. phase	Communication (bits)	Rounds
[HSS17]	$(3B^2+1)(n-1)\|\mathcal{C}\|(\kappa+\rho)+(n-1)\|\mathcal{I}\|(\kappa+\rho)$	13
[WRK17b]	$(3B+\frac{\kappa}{\kappa+\rho}B+1)(n-1)\|\mathcal{C}\|(\kappa+\rho)+(n-1)\|\mathcal{I}\|(\kappa+\rho)$	9
This paper	$(4B+1)(n-1)\|\mathcal{C}\|\kappa+(1-1/n)\|\mathcal{I}\|\kappa$	8
Func. Dep. phase	Communication (bits)	Rounds
[HSS17]	$4n\|\mathcal{C}\|\kappa+2(n-1)\|\mathcal{C}\|+(n-1)\|\mathcal{I}\|+\|\mathcal{O}\|$	2
[WRK17b]	$4(n-1)\|\mathcal{C}\|\kappa+4\|\mathcal{C}\|\rho+2(n-1)\|\mathcal{C}\|+(n-1)\|\mathcal{I}\|$	2
This paper	$(4n-6)\|\mathcal{C}\|\kappa+(2n-1)\|\mathcal{C}\|$	2
Online phase	Communication (bits)	Rounds
[HSS17]	$\|\mathcal{I}\|(n-1)\kappa+\|\mathcal{I}\|/n$	2
[WRK17b]	$\|\mathcal{I}\|\kappa+\|\mathcal{I}\|/n+\|\mathcal{O}\|$	3
This paper	$\max\{\|\mathcal{I}\|\kappa,\|\mathcal{C}\|\}+\|\mathcal{I}\|/n+\|\mathcal{O}\|$	4

Table 2: Comparison of communication complexity and rounds between our MPC protocol and the state-of-the-art protocols. The communication complexity is the maximum amount of data sent by any one party per execution. The columns for $\#R$ denote the rounds. $|\mathcal{I}|$ (resp., $|\mathcal{O}|$) is the length of all circuit-input (resp., circuit-output) wires, and thus $|\mathcal{I}|/n$ is the length of every party's input. $|\mathcal{C}|$ is the number of AND gates in the circuit.

CO15, CSW20]. In this phase, protocol Π_{mpc} needs to compute $|\mathcal{I}|$ authenticated bits, $|\mathcal{C}|$ authenticated shares and $|\mathcal{C}|$ AND triples, and thus needs about $(4B+1)|\mathcal{C}|(n-1)\kappa+|\mathcal{I}|(1-1/n)\kappa$ bits of communication per execution for every party. In each execution of the function-dependent phase, our protocol needs two rounds for computing circuit-dependent AND triples and sending a distributed garbled circuit, and requires at most $(4n-6)|\mathcal{C}|\kappa+(2n-1)|\mathcal{C}|$ bits of communication per party and $(4n-6)(n-1)|\mathcal{C}|\kappa+2n(n-1)|\mathcal{C}|+|\mathcal{C}|$ bits in total. In the online phase, protocol Π_{mpc} requires 4 rounds for that all parties obtain their outputs, and needs about $|\mathcal{I}|\kappa+|\mathcal{I}|/n+|\mathcal{O}|$ (resp., $|\mathcal{C}|+|\mathcal{I}|/n+|\mathcal{O}|$) bits of communication per execution for every garbler P_i (resp., the evaluator P_1).

In Table 2, we compare our MPC protocol with the state-of-the-art constant-round protocols [HSS17, WRK17b], where all protocols are optimized by using the amortized opening procedure (as described in Appendix A.3) for authenticated bits/shares. Recall that $B=\frac{\rho}{\log|\mathcal{C}|+1}+1$ denotes the bucket size. The communication rounds and complexity are obtained from the work [HSS17] for HSS and calculated from the protocol description for WRK [WRK17b].

In the online phase, we assume that every party obtains a possible different output. It is straightforward to extend the HSS protocol [HSS17] for supporting multiple different outputs. No explicit approach to support multiple outputs is described for the WRK protocol in their work [WRK17b]. The main problem is how every garbler P_i with $i\neq 1$ obtains the correct public values on its circuit-output wires in an efficient way, where recall that only the evaluator P_1 can compute the circuit. We can solve the problem, by considering a garbled label $\mathsf{L}^i_{w,\Lambda_w}=\mathsf{L}^i_{w,0}\oplus\Lambda_w\Delta_i$ as an MAC on bit Λ_w. Thus, we can let P_1 send the public values along with a hash value of garbled labels on these values to every garbler P_i who can check the correctness of these values. In this way, we can extend the WRK protocol [WRK17b] to support multiple different outputs.

From Table 2, our protocol obtains lower communication complexity in the preprocessing phases, and

	Comm.	Running time with different threads (*ns*)				
	(bytes)	2	4	8	26	32
[WRK17a]	286	3978	2255	1263	765	588
This paper	193	677	412	239	203	184
Improvement	1.48×	5.88×	5.47×	5.28×	3.77×	3.2×

Table 3: **Comparison of our protocol and WRK in the two-party setting.** The running time, which is needed to generated one authenticated AND triple, is reported in nanoseconds (*ns*). Communication cost is the amount of bandwidth needed per party to compute one authenticated triple.

has the (almost) same online communication overhead. Although our protocol has more rounds in the online phase, we believe that this is a reasonable trade-off for lower communication cost in the function-dependent phase. We refer the reader to Section 5 for the comparison of the concrete communication cost and performance.

5 Performance Evaluation

In this section, we compare the performance of our protocol with the best prior work. We developed an automatic benchmarking platform to remotely control a large number of machines executing MPC without the need to log in each machine. We will make it publicly available for all implementations that we produced from this work, as well as this testing platform. The code of the implementation is available to the reviewer upon request. For all protocols, we choose computational security parameter $\kappa = 128$ and statistical security parameter $\rho = 40$. All experiments are executed across machines of type `c5.9xlarge` with 36 vCPUs. The network bandwidth is 10 Gbps with latency about 0.1 ms.

5.1 Improvements for Authenticated Triple Generation Protocols

Two-party setting. In Table 3, we compare our authenticated triple generation protocol with the best prior implementation available by Wang et al. [WRK17a]. We compare the performance by using both implementations to compute 2^{23} authenticated triples and report the number of nanoseconds per triple. To demonstrate the computation-communication cost, we run the same experiment with different number of threads. For a fair comparison, we applied the same code optimization that we did in our code to the original WRK code. As a result, our reported WRK performance is actually twice faster than the performance reported in their paper. However, even after all these extra optimizations are applied to WRK, our protocol is still 5× faster than WRK when eight or less number of threads are used. When the number of threads approaches 32, we observe that the improvement decreases to 3×. This is because the network gradually becomes the bottleneck and limits the performance of our protocol.

Multi-party setting. In Figure 8 (the case of small number of parties) and Figure 9 (the case of large number of parties), we compare our authenticated triple generation protocol with the best prior implementation [WRK17b] in the multi-party setting. Similar to the two-party setting, we applied all our code optimizations to the original WRK protocol [WRK17b] so that the comparison does not include pure engineering effort. We keep the number of threads used in both cases and two protocols the same: for n-party triple generation, the number of threads used is $2(n-1)$. We observe higher fluctuation in running time for the multi-party setting, especially when n is large. Therefore, we also include the standard derivation as the error bar.

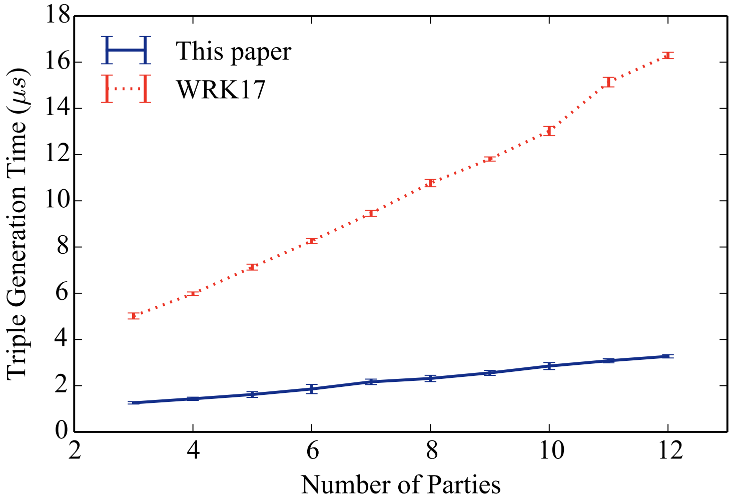

Figure 8: **Comparing our protocol with WRK for generating a multi-party authenticated AND triple (small number of parties).** The cost of generating one triple in the multi-party setting with number of parties from 3 to 12. Error bars show the standard derivation.

The WRK implementation frequently hangs when computing across more than 50 parties. It runs fine when the triple is less than 2^{15} where the bucket size is 4. To make it fair, we only compare up to the number of parties when WRK can run smoothly over 2^{20} triples. We can observe that our protocol consistently improves the efficiency by at least $5\times$. What's more, the running time of WRK for 16 parties is already slower than our protocol executed over 80 parties! We also give the exact running time for both protocols in Table 4 for different number of parties.

5.2 Improvements for Authenticated Garbling

Our improvements in the triple generation protocol directly translate to improvements of the function-independent phase in our MPC protocol. Here, we will mainly describe our improvements in the function-dependent phase for the authenticated garbling protocol. Our improvements mostly focus on the communication complexity in the multi-party setting, which also reflect the overall running time since the computation is cheap. Therefore, we will compare the communication complexity in the multi-party setting with the best known MPC protocol [WRK17b].

To make the comparison fair, we optimize the size of distributed garbled circuit of WRK [WRK17b] by using the trick proposed in the two-party setting [WRK17a]. This reduces the size of garbled circuit for WRK [WRK17b] from $4n\kappa$ bits per AND gate to $4(n-1)\kappa + 4\rho$ bits per AND gate. In addition, the online communication cost of WRK is obtained by using the amortized opening of authenticated bits.

Comparison of communication cost based on AES circuit. An AES circuit consists of 6800 AND gates and 128 bits of input and output. In the multi-party setting, we assume that all parties hold XOR shares of the input and the circuit will first XOR all input shares before the AES computation. Table 5 compares the communication cost for secure AES evaluation between our protocol and the best prior protocol [WRK17b] in the multi-party setting.

Compared to WRK [WRK17b], our protocol gives about $1.52\times$ improvement for three-party case and $1.27\times$ improvement for five-party case in the function-dependent phase. In terms of the communication cost of function-independent phase, our protocol leads to a $1.3\times$ improvement with a single execution and $1.26\times$ improvement with 1024 executions. While reducing the communication in both preprocessing phases, we

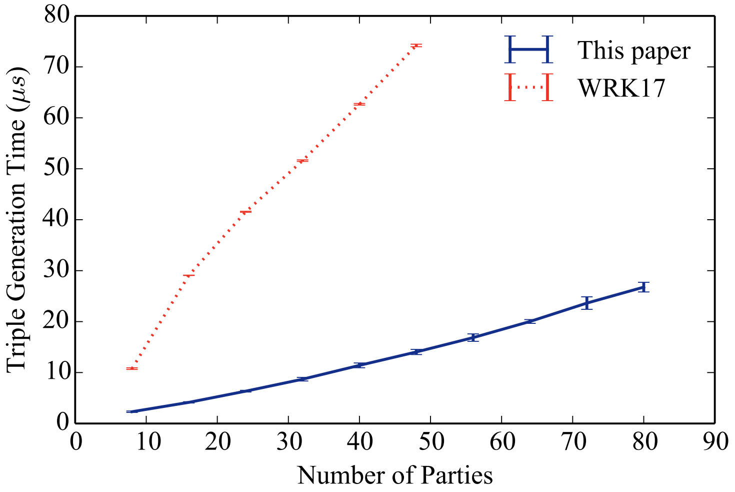

Figure 9: **Comparing our protocol with WRK for generating a multi-party authenticated AND triple (large number of parties).** The cost of generating one triple in the multi-party setting with number of parties from 8 to 80. Error bars show the standard derivation.

#Parties	3	4	5	6	7	8
WRK [WRK17b]	5.02	5.98	7.13	8.26	9.46	10.77
This paper	1.26	1.43	1.62	1.86	2.17	2.31
Improvement	3.98×	4.12×	4.4×	4.44×	4.36×	4.66×
#Parties	16	24	32	40	48	56
WRK [WRK17b]	29.09	40.55	51.6	62.66	75	–
This paper	4.95	6.37	8.7	11.41	13.54	16.86
Improvement	5.88×	6.36×	5.9×	5.5×	5.54×	–

Table 4: **Comparison of our protocol and WRK in the multi-party setting.** The running time, which is needed to generated one authenticated AND triple, is reported in microsecond (μs).

#Parties	Protocol	Func. Ind. (MB)		Func. Dep.	Online
		#1	#1024	(MB)	(KB)
$n = 3$	WRK [WRK17b]	4.8	3.6	1.0	6.3
	Ours	3.7	2.8	0.66	6.2
$n = 5$	WRK [WRK17b]	9.7	7.2	1.9	10.4
	Ours	7.5	5.7	1.5	10.3

Table 5: **Comparison of communication cost between our MPC protocol and the best known protocol for secure AES evaluation.** All numbers are the maximum amount of data sent by any one party per execution. The columns for "#1" and "#1024" denote the communication cost over a single execution and the amortized cost over 1024 executions respectively.

do not increase the communication cost in the online phase. In terms of the total communication from three phases, our protocol results in about 1.3× improvement for both a single execution and 1024 executions.

#Parties	Protocol	Func. Ind. (MB)	Func. Dep. (MB)	Online (MB)
$n = 3$	WRK [WRK17b]	1352.2	311.4	101.5
	Ours	942.2	202.6	100.9
$n = 5$	WRK [WRK17b]	3056.6	580.9	169.1
	Ours	1884.3	472.1	168.0

Table 6: **Communication cost of our and prior best protocols for computing Hamming distance.**

#Parties	Protocol	Func. Ind. (MB)	Func. Dep. (MB)	Online (MB)
$n = 3$	WRK [WRK17b]	5319.7	1518.1	6.4
	Ours	4269.8	987.8	6.3
$n = 5$	WRK [WRK17b]	10661.5	2831.8	10.6
	Ours	8539.5	2301.5	10.5

Table 7: **Communication cost of our and prior best protocols for secure sorting evaluation.**

Comparison of communication cost based on other circuits. In Table 6 and Table 7, we also compare our protocol with the state-of-the-art protocol [WRK17b] for circuits of other shapes, including *hamming distance* and *sorting*. As described in [WRK17a], these two circuits provide the following functionality:

- **Hamming distance.** In the multi-party setting, every party inputs an XOR-share of two bit-strings of length 1048576 bits. The circuit first XORs the shares to recover the underlying two bit-strings, and then output a 22-bit number containing the hamming distance of the two strings. The circuit includes 2097K AND gates.

- **Sorting.** Each party inputs an XOR-share of 4096 32-bit numbers. The circuit first XORs them to recover the numbers, and then sorts the numbers. The sorting circuit has 10223K AND gates.

Here, we only compare the communication cost of a single execution, as the circuits are large enough to take advantage of amortization within the circuit. In the function-independent phase of secure Hamming distance evaluation, our optimizations result in $1.44\times$ and $1.62\times$ improvements for three-party and five-party cases respectively. For sorting circuit, our protocol gives a $1.25\times$ improvement in the function-independent phase. In the function-dependent phase, our protocol gets a $1.54\times$ improvement in the communication for three-party case, and a $1.23\times$ improvement for five-party case. In particular, compared to WRK, our protocol reduces the total communication by more than 500 MB when $n = 3$ (1 GB for $n = 5$) for secure Hamming distance evaluation and 1.5 GB when $n = 3$ (2.5 GB for $n = 5$) for sorting.

References

[ALSZ13] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *ACM Conf. on Computer and Communications Security (CCS) 2013*, pages 535–548. ACM Press, 2013.

[AOR$^+$19] Abdelrahaman Aly, Emmanuela Orsini, Dragos Rotaru, Nigel P. Smart, and Tim Wood. Zaphod: Efficiently combining lsss and garbled circuits in scale. In *Proceedings of the 7th ACM*

Workshop on Encrypted Computing & Applied Homomorphic Cryptography, WAHC'19, page 33–44, 2019.

[BCG$^+$19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *ACM Conf. on Computer and Communications Security (CCS) 2019*, pages 291–308. ACM Press, 2019.

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

[Bea92] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology—Crypto 1991*, LNCS, pages 420–432. Springer, 1992.

[BJPR18] Megha Byali, Arun Joseph, Arpita Patra, and Divya Ravi. Fast secure computation for small population over the internet. In *ACM Conf. on Computer and Communications Security (CCS) 2018*, pages 677–694. ACM Press, 2018.

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

[BLO17] Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Efficient scalable constant-round MPC via garbled circuits. In *Advances in Cryptology—Asiacrypt 2017, Part II*, LNCS, pages 471–498. Springer, 2017.

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

[CDD$^+$16] Ignacio Cascudo, Ivan Damgård, Bernardo David, Nico Döttling, and Jesper Buus Nielsen. Rate-1, linear time and additively homomorphic UC commitments. In *Advances in Cryptology—Crypto 2016, Part III*, volume 9816 of *LNCS*, pages 179–207. Springer, 2016.

[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.

[CO15] Tung Chou and Claudio Orlandi. The simplest protocol for oblivious transfer. In *Progress in Cryptology—Latincrypt 2015*, LNCS, pages 40–58. Springer, 2015.

[CSW20] Ran Canetti, Pratik Sarkar, and Xiao Wang. Blazing fast ot for three-round uc ot extension. Cryptology ePrint Archive, Report 2020/110, to appear at PKC'20, 2020. `https://eprint.iacr.org/2020/110`.

[DEF$^+$19] Ivan Damgård, Daniel Escudero, Tore Kasper Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. New primitives for actively-secure MPC over rings with applications to private machine learning. In *IEEE Symposium on Security and Privacy (S&P) 2019*, pages 1102–1120, 2019.

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

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

[GKWY19] Chun Guo, Jonathan Katz, Xiao Wang, and Yu Yu. Efficient and secure multiparty computation from fixed-key block ciphers. Cryptology ePrint Archive, Report 2019/074, 2019. https://eprint.iacr.org/2019/074.

[GL05] Shafi Goldwasser and Yehuda Lindell. Secure multi-party computation without agreement. *J. 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 *19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229. ACM Press, 1987.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004.

[HIV17] Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Actively secure garbled circuits with constant communication overhead in the plain model. In *Theory of Cryptography Conference (TCC) 2017*, LNCS, pages 3–39. Springer, 2017.

[HSS17] Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. In *Advances in Cryptology—Asiacrypt 2017, Part I*, LNCS, pages 598–628. Springer, 2017.

[HVW19] Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, and Mor Weiss. The price of active security in cryptographic protocols. Cryptology ePrint Archive, Report 2019/1250, 2019. https://eprint.iacr.org/2019/1250.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology—Crypto 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, 2003.

[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.

[KRRW18] Jonathan Katz, Samuel Ranellucci, Mike Rosulek, and Xiao Wang. Optimizing authenticated garbling for faster secure two-party computation. In *Advances in Cryptology—Crypto 2018, Part III*, volume 10993 of *LNCS*, pages 365–391. Springer, 2018.

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

[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.

[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 *Theory of Cryptography Conference (TCC) 2016*, LNCS, pages 554–581. Springer, 2016.

[Nie07] Jesper Buus Nielsen. Extending oblivious transfers efficiently - how to get robustness almost for free. Cryptology ePrint Archive, Report 2007/215, 2007. http://eprint.iacr.org/2007/215.

[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.

[NST17] Jesper Buus 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.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *Advances in Cryptology—Crypto 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, 2008.

[Rin] Peter Rindal. libOTe: an efficient, portable, and easy to use Oblivious Transfer Library. https://github.com/osu-crypto/libOTe.

[RW19] Dragos Rotaru and Tim Wood. MArBled circuits: Mixing arithmetic and Boolean circuits with active security. LNCS, pages 227–249. Springer, 2019.

[WRK17a] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In *ACM Conf. on Computer and Communications Security (CCS) 2017*, pages 21–37. ACM Press, 2017.

[WRK17b] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In *ACM Conf. on Computer and Communications Security (CCS) 2017*, pages 39–56. ACM Press, 2017.

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

[ZCSH18] Ruiyu Zhu, Darion Cassel, Amr Sabry, and Yan Huang. NANOPI: Extreme-scale actively-secure multi-party computation. In *ACM Conf. on Computer and Communications Security (CCS) 2018*, pages 862–879. ACM Press, 2018.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *Advances in Cryptology—Eurocrypt 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, 2015.

A More Background

A.1 Commitment and Coin-tossing

Our protocols need two standard functionalities for commitment and coin-tossing respectively.

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

This functionality runs with parties $P_1, \ldots, P_n$ as follows:

Commit: On input $(\mathsf{Commit}, i, x, \tau_x)$ from P_i, store (i, x, τ_x) and output (i, τ_x) to all parties.

Open: On input $(\mathsf{Open}, i, \tau_x)$ from P_i, if a tuple (i, x, τ_x) was previously stored, output (i, x, τ_x) to all parties. If instead $(\mathsf{NoOpen}, i, \tau_x)$ is given by the adversary and P_i is corrupted, output $(i, \perp, \tau_x)$ to all parties.

Figure 10: Functionality for commitments.

Commitment. We will use a commitment functionality shown in Figure 10. This functionality can easily be implemented in the random oracle model [HSS17] via defining $\mathsf{Commit}(i, x) = H(i, x, r)$ where $H : \{0,1\}^* \to \{0,1\}^{2\kappa}$ is a random oracle and $r \in \{0,1\}^\kappa$ is a randomness, where note that $\mathsf{Commit}(i, x)$ needs to be broadcast in the multi-party setting.

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

This functionality runs with n parties $P_1, \ldots, P_n$ as follows:

- Upon receiving $(\mathsf{Rand}, \mathcal{R})$ from all parties where $\mathcal{R}$ is any efficiently sampleable set, sample $r \leftarrow_{\$} \mathcal{R}$ and send r to all parties.

Figure 11: Coin-tossing functionality.

Coin tossing. We will use a standard coin-tossing functionality $\mathcal{F}_{\mathsf{Rand}}$ shown in Figure 11, which samples an entry from any efficiently sampleable set $\mathcal{R}$. This can be securely realized in the random oracle model by having every party commit to a random $\mathsf{seed}_i \in \{0,1\}^\kappa$ via calling $\mathcal{F}_{\mathsf{Com}}$, and then open all commitments and use $\bigoplus_{i \in [n]} \mathsf{seed}_i$ as a seed to sample an element from set $\mathcal{R}$.

A.2 Almost Universal Linear Hash Functions

We will use a family of almost universal linear hash functions [CDD+16] over $\mathbb{F}_{2^s}$ for some parameter $s \in \mathbb{N}$, which is defined as follows:

Definition 1 (Almost Universal Linear Hashing). *We say that a family $\mathcal{H}$ of linear hash functions $\mathbb{F}_{2^s}^m \to \mathbb{F}_{2^s}$ is ϵ-almost universal, if it holds for every non-zero vector $\mathbf{x} \in \mathbb{F}_{2^s}^m$ such that*

$$\Pr_{\mathbf{H} \leftarrow \mathcal{H}} [\mathbf{H}(\mathbf{x}) = 0] \leq \epsilon,$$

where $\mathbf{H}$ is chosen uniformly at random from the family $\mathcal{H}$.

Efficient constructions for a family of almost universal linear hash functions have been proposed such as [DPSZ12, CDD+16, NST17]. In this paper, we adopt the following practical construction, which is a polynomial hash based on GMAC and also used in [NST17, HSS17]:

- Sample a random seed $\chi \leftarrow \mathbb{F}_{2^s}$.
- Use χ to define the following linear hash function $\mathbf{H}$:

$$\mathbf{H} : \mathbb{F}_{2^s}^m \to \mathbb{F}_{2^s}, \qquad \mathbf{H}(x_1, x_2, \ldots, x_m) = x_1 \cdot \chi + x_2 \cdot \chi^2 + \cdots + x_m \cdot \chi^m$$

The seed $\chi \in \mathbb{F}_{2^s}$ is short, but the computational complexity is $O(m \cdot s)$. When $s = 128$ is adopted, the finite field multiplication over $\mathbb{F}_{2^s}$ can be performed very efficiently in hardware on modern CPUs by using the Intel SSE instruction [NST17]. This construction described as above provides an almost universal family with $\epsilon = m \cdot 2^{-s}$, as χ is uniformly random in $\mathbb{F}_{2^s}$ and independent of the input $\mathbf{x} = (x_1, x_2, \ldots, x_m)$. This can be improved to 2^{-s}, at the cost of a larger seed, by using m different random coefficients.

A.3 Amortized Opening Procedures

In this section, we present how to open authenticated bits/shares in an amortized way (i.e., it is possible to open ℓ authenticated bits with less than ℓ times the communication) using the standard techniques [NNOB12, DPSZ12]. In a naive approach, a party P_i can open $[x]_i^j$ to P_j via just sending x and $\mathsf{M}_j[x]$ to P_j. Party P_j is able to verify the validity of x by checking that $\mathsf{M}_j[x] = \mathsf{K}_j[x] \oplus x\Delta_j$. As observed in previous work [NNOB12], authenticated bits/shares can be opened in the following amortized process.

- **aBits:** For each $i \in [n], j \neq i$, P_i can open ℓ two-party authenticated bits $[x_1]_i^j, \ldots, [x_\ell]_i^j$ to P_j as follows:

  1. P_i sends $x_1, \ldots, x_\ell$ and $\tau_j := \mathsf{H}(\mathsf{M}_j[x_1], \ldots, \mathsf{M}_j[x_\ell])$ to P_j.
  2. P_j checks that $\tau_j = \mathsf{H}(\mathsf{K}_j[x_1] \oplus x_1\Delta_j, \ldots, \mathsf{K}_j[x_\ell] \oplus x_\ell\Delta_j)$. If the check fails, P_j aborts.

  We use $\mathsf{Open}([x_k]_i^j)$ for each $k \in [\ell]$ to denote the above amortized opening process for two-party authenticated bits. P_i can also open ℓ multi-party authenticated bits $[x_1]_i, \ldots, [x_\ell]_i$ to all parties via opening $[x_1]_i^j, \ldots, [x_\ell]_i^j$ to P_j for each $j \neq i$.

- **aShares:** All parties can open ℓ authenticated shares $\langle x_1 \rangle, \ldots, \langle x_\ell \rangle$ by every party P_i opening its portion in the following way.

  1. For each $j \neq i$, P_i sends $x_1^i, \ldots, x_\ell^i$ along with $\tau_{i,j} := \mathsf{H}(\mathsf{M}_j[x_1^i], \ldots, \mathsf{M}_j[x_\ell^i])$ to P_j.
  2. For $j \neq i$, P_j checks that $\tau_{i,j} = \mathsf{H}(\mathsf{K}_j[x_1^i] \oplus x_1^i\Delta_j, \ldots, \mathsf{K}_j[x_\ell^i] \oplus x_\ell^i\Delta_j)$, and aborts if the check fails.

  Let $\mathsf{Open}(\langle x_k \rangle)$ for each $k \in [\ell]$ denote the above amortized opening process for authenticated shares.

Below, we prove that the above opening process in a batch is secure in the random oracle model, even if the adversary can leak a few bits of global keys such that each bit leaked of global keys will be caught with probability $1/2$. We focus on the case of two-party authenticated bits, where the security proof is easy to be generalized to multi-party authenticated bits and authenticated shares.

Lemma 1. *If $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\kappa$ is a random oracle, in the amortized opening process for two-party authenticated bits, either an honest party P_j aborts, or P_j receives the correct bits from a malicious party P_i except with probability $(q+1)/2^\kappa$, where q is an upper bound of the number of queries to H. Let $\mathcal{A}$ be a probabilistic polynomial time (PPT) adversary, which corrupts the party P_i. Assume that $\mathcal{A}$ leaks c bits of global key Δ_j for some $c \in [\kappa] \cup \{0\}$, and honest party P_j will abort with probability $1/2^c$.*

Proof. Let $x_1, \ldots, x_\ell$ be the correct bits that will be sent by semi-honest P_i. In the opening process, adversary $\mathcal{A}$ on behalf of P_i sends the bits $x'_1, \ldots, x'_\ell$ along with τ'_j to honest party P_j. If P_j does not abort, then $\tau'_j = \mathsf{H}(\mathsf{K}[x_1] \oplus x'_1\Delta_j, \ldots, \mathsf{K}[x_\ell] \oplus x'_\ell\Delta_j)$. If $\mathcal{A}$ makes a query $\mathbf{z}$ to H such that $\mathsf{H}(\mathbf{z}) = \tau'_j$ but

$\mathbf{z} \neq (\mathsf{K}[x_1] \oplus x'_1\Delta_j, \ldots, \mathsf{K}[x_\ell] \oplus x'_\ell\Delta_j)$, then $\mathcal{A}$ finds a target collision for random oracle H, which happens with probability $q/2^\kappa$.

Below, we assume that $\mathcal{A}$ does not find a target collision, and then analyze the probability that there exists some $k \in [\ell]$ such that $x'_k \neq x_k$. The probability that $\mathcal{A}$ forges an information-theoretic MAC $\mathsf{M}_j[x'_k] = \mathsf{K}_j[x_k] \oplus x'_k\Delta_j$ is bounded by $1/2^{\kappa-c}$. Note that P_j will abort except with probability $1/2^c$, due to the c leaked bits of Δ_j. Together, the probability that P_j does not abort and $\mathcal{A}$ forges an MAC $\mathsf{M}_j[x'_k]$ is $1/2^c \cdot 1/2^{\kappa-c} = 1/2^\kappa$.

Overall, except with probability $(q+1)/2^\kappa$, P_j will receive the correct bits, if it does not abort. □

B Proof of Security for Our Authenticated Bit Protocol

B.1 Analysis of Checking in the aBit Protocol

Analysis of correlation check: For the security analysis of correlation check, we recall an important lemma by Keller et al. [KOS15]. Here we consider that P_i is corrupted by the adversary. Without loss of generality, we fix an honest party P_j to analyze the correlation check. When calling the (extend) command of functionality $\mathcal{F}_{\mathsf{COTe}}$, a corrupt party P_i may send a vector $\mathbf{x}_{k,j}$ for $k \in [\ell']$ to $\mathcal{F}_{\mathsf{COTe}}$, and receives an MAC $\mathsf{M}'_j[x_{k,j}] := \mathsf{K}_j[x_{k,j}] + \mathbf{x}_{k,j} * \Delta_j$ for $k \in [\ell']$. We take P_i's inputs $\mathbf{x}_{1,j}, \ldots, \mathbf{x}_{\ell',j} \in \mathbb{F}_2^\kappa$ to be the rows of an $\ell' \times \kappa$ matrix. Let $\hat{\mathbf{x}}_{j,1}, \ldots, \hat{\mathbf{x}}_{j,\kappa} \in \mathbb{F}_2^{\ell'}$ be the columns of the same matrix. If P_i is semi-honest, then $\mathbf{x}_{k,j}$ for $k \in [\ell']$ is monochrome, and $\hat{\mathbf{x}}_{j,1}, \ldots, \hat{\mathbf{x}}_{j,\kappa}$ are all equal. Given a sender P_j and a receiver P_i, our correlation check for two parties without broadcast is the same as that by Keller et al. [KOS15]. Thus, we can use the following lemma by Keller et al. [KOS15] to prove the security of the correlation check in our protocol Π_{aBit}.

Lemma 2 ([KOS15])**.** *Let $S_{\Delta_j} \subseteq \mathbb{F}_2^\kappa$ be the set of all Δ_j for which the correlation check passes, given the view of receiver P_i. Except with probability $2^{-\kappa}$, there exists $d_j \in \mathbb{N}$ such that*

1. $|S_{\Delta_j}| = 2^{d_j}$.
2. *For each $\mathbf{s} \in \{\hat{\mathbf{x}}_{j,l}\}_{l\in[\kappa]}$, let $H_\mathbf{s} = \{l \in [\kappa] \mid \mathbf{s} = \hat{\mathbf{x}}_{j,l}\}$. Then one of the following holds:*
   - *For all $l \in H_\mathbf{s}$ and any $\Delta_j^{(1)}, \Delta_j^{(2)} \in S_{\Delta_j}$, $\Delta_j^{(1)}[l] = \Delta_j^{(2)}[l]$.*
   - *$|H_\mathbf{s}| \geq d_j$ and $\left|\{\Delta_j[H_\mathbf{s}]\}_{\Delta_j \in S_{\Delta_j}}\right| = 2^{d_j}$, where $\Delta_j[H_\mathbf{s}]$ denotes the vector consisting of the bits $\{\Delta_j[l]\}_{l\in H_\mathbf{s}}$. In other words, S_{Δ_j} restricted to the bits corresponding to $H_\mathbf{s}$ has entropy at least d_j. Furthermore, there exists $\hat{\mathbf{s}}$ such that $|H_{\hat{\mathbf{s}}}| \geq d_j$.*

According to the analysis by Keller et al. [KOS15], we give some intuition about the above lemma. The probability of passing the correlation check is $|S_{\Delta_j}|/2^\kappa$, as Δ_j is sampled uniformly at random by P_j. For a semi-honest P_i, $H_\mathbf{s}$ is always the set $\{1, \ldots \kappa\}$. So the size of $H_\mathbf{s}$ reflects the number of deviation in the protocol for a given $\mathbf{s}$. Furthermore, the precise indices in $H_\mathbf{s}$ correspond to a subset of the bits of Δ_j. The second part of Lemma 2 implies that for any $\mathbf{s}$, either the bits of Δ_j corresponding to the indices in $H_\mathbf{s}$ are known, or the size of $H_\mathbf{s}$ is at least d_j. In the first case, the bits of Δ_j are revealed by the adversary corrupting P_i by guessing the bits and observing whether the correlation check passes. In the second case, we have a bound on the amount of information that the adversary can learn. In particular, the total amount of the bits of Δ_j learned by the adversary is bounded by $c_j = \kappa - d_j$, since $|S_{\Delta_j}| = 2^{d_j}$ and S_{Δ_j} restricted to the bits corresponding to $H_{\hat{\mathbf{s}}}$ has entropy at least d_j.

Let $x_{1,j}, \ldots, x_{\ell',j}$ be the bits in vector $\hat{\mathbf{s}}$. Then, for $k \in [\ell']$, we can write the MAC with an error received by the malicious party P_i as $\mathsf{M}'_j[x_{k,j}] = \mathsf{K}_j[x_{k,j}] + x_{k,j} \cdot \Delta_j + \mathbf{e}_{k,j} * \Delta_j$, where $\mathbf{e}_{k,j} = (x_{k,j}, \ldots, x_{k,j}) + \mathbf{x}_{k,j} \in$

$\mathbb{F}_2^\kappa$ is an adversarially chosen error vector. For each $k \in [\ell']$, by the definition of $H_{\mathbf{s}}$ and $\mathbf{e}_{k,j}$, we have that $\mathbf{e}_{k,j}[l] = \mathbf{e}_{k,j}[l']$ for all $l, l' \in H_{\mathbf{s}}$, for any $\mathbf{s} \in \{\hat{\mathbf{x}}_{j,1}, \dots, \hat{\mathbf{x}}_{j,\kappa}\}$. Note that $\mathbf{e}_{k,j}[l] = 0$ for all $l \in H_{\hat{\mathbf{s}}}$, as $\mathbf{x}_{k,j}[l] = x_{k,j}$ for all $l \in H_{\hat{\mathbf{s}}}$. This implies that $\mathbf{e}_{k,j}[l] \cdot \Delta_j[l] = 0$ for all $l \in H_{\hat{\mathbf{s}}}$. Lemma 2 implies that there exists only one $\mathbf{s} = \hat{\mathbf{s}}$ such that the second case happens, except with probability $2^{-\kappa}$. [2] That is, for $\mathbf{s} \neq \hat{\mathbf{s}}$, the first case occurs in Lemma 2 except with probability $2^{-\kappa}$. In this case, for all $k \in [\ell']$ and $l \in H_{\mathbf{s}}$, $\mathbf{e}_{k,j}[l] \cdot \Delta_j[l]$ is known by the adversary by the fact that $\Delta_j \in S_{\Delta_j}$. Therefore, for $k \in [\ell']$, the adversary knows $\mathbf{e}_{k,j} * \Delta_j$ and thus the correct MAC $\mathsf{M}_j[x_{k,j}] = \mathsf{K}_j[x_{k,j}] + x_{k,j} \cdot \Delta_j$. In addition, we will use the following lemma.

Lemma 3 ([KOS15])**.** *Let $\mathbf{A}$ be a random $(t+m) \times t$ matrix over $\mathbb{F}_2$ where $m > 0$. Then $\mathbf{A}$ has rank t except with probability less than 2^{-m}.*

Analysis of consistency check: Now, we assume that the outputs have the correct correlation, i.e., $\mathsf{M}_j[x_{k,j}] = \mathsf{K}_j[x_{k,j}] + x_{k,j} \cdot \Delta_j$ for all $k \in [\ell'], j \neq i$. When calling the (extend) command of $\mathcal{F}_{\mathsf{COTe}}$, the malicious party P_i may use inconsistent inputs x_k for $k \in [\ell']$ with two different honest parties. In particular, we define $\{x_{k,j}\}_{k \in [\ell']}$ to be the actual bits used by P_i when calling $\mathcal{F}_{\mathsf{COTe}}$ with an honest party P_j. Without loss of generality, we choose an honest party P_{j_0} and fix $x_k = x_{k,j_0}$ for each $k \in [\ell']$. For each $j \in \mathcal{H}$ and $k \in [\ell']$, $x_{k,j}$ can be denoted as $x_{k,j} = x_k + \delta_{k,j} \in \mathbb{F}_2$, where $\delta_{k,j_0} = 0$. Based on Lemma 2 and the above analysis, we prove that the malicious party P_i cannot use inconsistent values $x_{k,j}$ to different honest parties in the following lemma.

Lemma 4. *For a corrupt party P_i and every honest party $P_j \notin A$, P_i and P_j holds a secret sharing of $x_k \cdot \Delta_j$ for each $k \in [\ell']$. In other words, for each $k \in [\ell']$ and $j \notin A$, $\delta_{k,j} = 0$.*

Proof. For each $j \notin A$, we define the MAC of corrupt party P_i on value $\sum_{k=1}^{\ell'} \chi_k \cdot x_{k,j}$ as $\mathsf{M}_j[\mathbf{y}^i] = \sum_{k=1}^{\ell'} \chi_k \cdot \mathsf{M}_j[x_{k,j}]$, and the local key of honest party P_j on the same value as $\mathsf{K}_j[\mathbf{y}^i] = \sum_{k=1}^{\ell'} \chi_k \cdot \mathsf{K}_j[x_{k,j}]$. For each $j \notin A, k \in [\ell']$, we have that $\mathsf{M}_j[x_{k,j}] = \mathsf{K}_j[x_{k,j}] + x_{k,j} \cdot \Delta_j$ known by the adversary corrupting P_i. In Step 6 of protocol Π_{aBit}, P_i may broadcast an incorrect value $\tilde{\mathbf{y}}^i = \mathbf{y}^i + \mathbf{e}^i$ to other parties where $\mathbf{y}^i = \sum_{k=1}^{\ell'} \chi_k \cdot x_k$, and send an incorrect MAC $\widehat{\mathsf{M}}_j[\mathbf{y}^i] = \mathsf{M}_j[\mathbf{y}^i] + \mathsf{E}_{i,j}$ to every honest party P_j. If $P_j \notin A$ does not abort, then $\widehat{\mathsf{M}}_j[\mathbf{y}^i] = \mathsf{K}_j[\mathbf{y}^i] + \tilde{\mathbf{y}}^i \cdot \Delta_j$. Thus, we have:

$$\begin{aligned}
&\mathsf{M}_j[\mathbf{y}^i] + \mathsf{E}_{i,j} = \mathsf{K}_j[\mathbf{y}^i] + \mathbf{y}^i \cdot \Delta_j + \mathbf{e}^i \cdot \Delta_j \\
&\Leftrightarrow \mathsf{E}_{i,j} + \mathbf{y}^i \Delta_j + \mathbf{e}^i \Delta_j = \mathsf{M}_j[\mathbf{y}^i] + \mathsf{K}_j[\mathbf{y}^i] = \left(\sum_{k=1}^{\ell'} \chi_k \cdot x_{k,j} \right) \cdot \Delta_j \\
&\Leftrightarrow \mathsf{E}_{i,j} = \left(\mathbf{y}^i + \mathbf{e}^i + \sum_{k=1}^{\ell'} \chi_k \cdot (x_k + \delta_{k,j}) \right) \cdot \Delta_j \\
&\Leftrightarrow \mathsf{E}_{i,j} = \left(\mathbf{e}^i + \sum_{k=1}^{\ell'} \chi_k \cdot \delta_{k,j} \right) \cdot \Delta_j
\end{aligned}$$

For each $j \notin A$, a corrupt P_i has the following two possible ways to cheat P_j, but succeeds with negligible probability in both cases.

1. If $\mathsf{E}_{i,j} \neq 0$, then $(\mathbf{e}^i + \sum_{k=1}^{\ell'} \chi_k \cdot \delta_{k,j}) \neq 0$, and thus the adversary can learn Δ_j. The P_j's check passes with probability $|S_{\Delta_j}| \cdot 2^{-\kappa} + 2^{-\kappa} = 2^{d_j - \kappa} + 2^{-\kappa}$. Therefore, the probability, that honest party P_j does not abort and the adversary learns Δ_j, is $(2^{d_j-\kappa} + 2^{-\kappa}) \cdot 2^{-d_j} = 2^{-\kappa} + 2^{-(\kappa+d_j)} < 2^{-\kappa+1}$.

[2] One can easily prove if there are two different $\mathbf{s}, \mathbf{s}'$ satisfying the second case of Lemma 2, then the correlation check will *not* pass except with probability $2^{-\kappa}$.

2. If $\mathsf{E}_{i,j} = 0$, then $\mathbf{e}^i = \sum_{k=1}^{\ell'} \chi_k \cdot \delta_{k,j}$ unless $\Delta_j = 0$ with probability $2^{-\kappa}$. As $\delta_{k,j_0} = 0$ for each $k \in [\ell']$, this implies that $\mathbf{e}^i = 0$. Thus, for each $j \notin A \backslash \{j^0\}$, we have that $\sum_{k=1}^{\ell'} \chi_k \cdot \delta_{k,j} = 0$. The probability, that there exists some $k \in [\ell']$ such that $\delta_{k,j} \neq 0$, is at most $2^{-\kappa}$, as $\{\delta_{k,j}\}_{k \in [\ell']}$ are independent of $\{\chi_k\}_{k \in [\ell']}$ and $\chi_1, \ldots, \chi_{\ell'}$ are uniformly random.

Overall, with probability at least $1 - 4 \cdot 2^{-\kappa}$, this lemma holds. □

B.2 Proof of Theorem 1

Theorem 4 (Theorem 1, restated)**.** *Protocol Π_{aBit} shown in Figure 4 securely realizes functionality $\mathcal{F}_{\mathsf{aBit}}$ in the $(\mathcal{F}_{\mathsf{COTe}}, \mathcal{F}_{\mathsf{Rand}})$-hybrid model.*

Proof. Let $\mathcal{A}$ be a probabilistic polynomial time (PPT) adversary, who corrupts a subset of parties $A \subset [n]$. We construct a PPT simulator $\mathcal{S}$ that has access to the functionality $\mathcal{F}_{\mathsf{aBit}}$ and simulates the adversary's view. Simulator $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs before it aborts or terminates the simulation. We consider two cases of honest P_i and malicious P_i separately. In both cases, we prove that the real world is indistinguishable from the ideal world.

Description of simulator. $\mathcal{S}$ emulates functionalities $\mathcal{F}_{\mathsf{COTe}}$ and $\mathcal{F}_{\mathsf{Rand}}$, interacts with adversary $\mathcal{A}$ and simulates as follows.

Case 1 (honest party $P_i \notin A$):

1. For each $j \in A$, $\mathcal{S}$ emulates the functionality $\mathcal{F}_{\mathsf{COTe}}$, and receives Δ_j and $\mathsf{K}_j[x_1], \ldots, \mathsf{K}_j[x_{\ell'}]$ from $\mathcal{A}$. Then $\mathcal{S}$ sends these values to $\mathcal{F}_{\mathsf{aBit}}$.

2. For the call of $\mathcal{F}_{\mathsf{Rand}}$ from $\mathcal{A}$, $\mathcal{S}$ samples random $\chi_1, \ldots, \chi_{\ell'}$, and then sends them to $\mathcal{A}$.

3. Acting as honest party P_i, for each $j \in A$, $\mathcal{S}$ computes $\mathsf{K}_j[\mathbf{y}^i] := \sum_{k=1}^{\ell'} \chi_k \cdot \mathsf{K}_j[x_k]$, samples $\mathbf{y}^i \leftarrow \mathbb{F}_{2^\kappa}$, and sends $\mathbf{y}^i$ and $\mathsf{M}_j[\mathbf{y}^i] = \mathsf{K}_j[\mathbf{y}^i] + \mathbf{y}^i \cdot \Delta_j$ to $\mathcal{A}$. For each $l \notin A \backslash \{i\}$, $\mathcal{S}$ samples a random $\mathsf{M}_l[\mathbf{y}^i] \leftarrow \mathbb{F}_{2^\kappa}$ and sends it to dummy party P_l.

Case 2 (corrupt party $P_i \in A$):

1. For each corrupt party $j \in A \backslash \{i\}$, $\mathcal{S}$ receives Δ_j from $\mathcal{A}$ for $\mathcal{F}_{\mathsf{COTe}}$, and then sends Δ_j to $\mathcal{F}_{\mathsf{aBit}}$. For each $j \notin A$, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{COTe}}$ and receives $\mathbf{x}_{1,j}, \ldots, \mathbf{x}_{\ell',j}$ and $\mathsf{M}'_j[x_{1,j}], \ldots, \mathsf{M}'_j[x_{\ell',j}]$ from $\mathcal{A}$ acting as corrupt party P_i.

2. For each $j \notin A$, let $\hat{\mathbf{s}}$ and $H_{\hat{\mathbf{s}}}$ be as in Lemma 2, i.e., $|H_{\hat{\mathbf{s}}}| \geq d_j$ and $\hat{\mathbf{x}}_{j,l} = \hat{\mathbf{x}}_{j,l'}$ for all $l, l' \in H_{\hat{\mathbf{s}}}$. This implies $\mathbf{x}_{k,j}[l] = \mathbf{x}_{k,j}[l']$ for all $l, l' \in H_{\hat{\mathbf{s}}}$ and $k \in [\ell']$. For each $k \in [\ell']$, $\mathcal{S}$ sets $x_{k,j} := \mathbf{x}_{k,j}[l]$ for some $l \in H_{\hat{\mathbf{s}}}$. For each $j \notin A$, $\mathcal{S}$ computes $\mathbf{e}_{k,j} := (x_{k,j}, \ldots, x_{k,j}) + \mathbf{x}_{k,j}$ for $k \in [\ell']$. Simulator $\mathcal{S}$ defines a set $S_j = \{l \in [\kappa] \mid \exists k \in [\ell'] \text{ s.t. } \mathbf{e}_{k,j}[l] = 1\}$ and sets $c_j := |S_j|$.

3. Upon receiving (Rand, ℓ') from $\mathcal{A}$, $\mathcal{S}$ emulates the functionality $\mathcal{F}_{\mathsf{Rand}}$, samples $\chi_k \leftarrow \mathbb{F}_{2^\kappa}$ for each $k \in [\ell']$, and sends these random coefficients to $\mathcal{A}$.

4. For each $j \notin A$, $\mathcal{S}$ computes $\mathbf{y}^{i,j} := \sum_{k=1}^{\ell'} \chi_k \cdot x_{k,j}$, and receives $\tilde{\mathbf{y}}^i$ from $\mathcal{A}$ over a broadcast channel. Then, $\mathcal{S}$ computes $\mathbf{e}^{i,j} := \tilde{\mathbf{y}}^i + \mathbf{y}^{i,j}$. If $\mathbf{e}^{i,j} \neq 0$, $\mathcal{S}$ aborts. Additionally, $\mathcal{S}$ receives $\widehat{\mathsf{M}}_j[\mathbf{y}^i]$ from $\mathcal{A}$, and then computes $E_j := \widehat{\mathsf{M}}_j[\mathbf{y}^i] + \sum_{k=1}^{\ell'} \chi_k \cdot \mathsf{M}'_j[x_{k,j}] \in \mathbb{F}_{2^\kappa}$. If $S_j = \emptyset$, $\mathcal{S}$ aborts if $E_j \neq 0$, and sets $\mathbf{e}_{k,j} * \Delta_j = 0$ for $k \in [\ell']$ otherwise. Otherwise, $\mathcal{S}$ continues the simulation.

5. For each $j \notin A$, if $S_j \neq \emptyset$, $\mathcal{S}$ can re-write $\sum_{k=1}^{\ell'} \chi_k \cdot (\mathbf{e}_{k,j} * \Delta_j)$ as $\mathbf{X}_j \cdot \mathbf{t}_j$, where $\mathbf{X}_j \in \mathbb{F}_2^{\kappa \times |S_j|}$ is a matrix determined by $\{\mathbf{e}_{k,j}\}_{k \in [\ell']}$ and $\{\chi_k\}_{k \in [\ell']}$, and $\mathbf{t}_j \in \mathbb{F}_2^{|S_j|}$ is a column vector such that $\mathbf{t}_j[l] = \Delta_j[l]$ for each $l \in S_j$. Then, $\mathcal{S}$ establishes the equation $\mathbf{X}_j \cdot \mathbf{t}_j = E_j$, and does the following:
   - If there is no solutions for the equation, $\mathcal{S}$ aborts.
   - If there is a unique solution for the equation (i.e., $\mathbf{X}_j$ has rank $c_j = |S_j|$), $\mathcal{S}$ computes the solution $\mathbf{t}_j$, and thus obtains a guess $\{\Delta'_j[l]\}_{l \in S_j}$ from $\mathcal{A}$.
   - If there are at least two solutions for the equation, $\mathcal{S}$ aborts.
6. For each $j \notin A$, if $S_j \neq \emptyset$, $\mathcal{S}$ sends $(\mathsf{leak}, j, S_j, \{\Delta'_j[l]\}_{l \in S_j})$ to $\mathcal{F}_{\mathsf{aBit}}$. If $\mathcal{S}$ receives $\mathtt{fail}$ from $\mathcal{F}_{\mathsf{aBit}}$, $\mathcal{S}$ aborts. Otherwise, $\mathcal{S}$ receives $\mathtt{success}$ from $\mathcal{F}_{\mathsf{aBit}}$ and is confirmed $\Delta_j[l] = \Delta'_j[l]$ for each $l \in S_j$.
7. For each $j \notin A$, if $S_j \neq \emptyset$, $\mathcal{S}$ computes $\mathbf{e}_{k,j} * \Delta_j$ for each $k \in [\ell']$, where $\mathbf{e}_{k,j}[l] \cdot \Delta_j[l] = 0$ for all $l \in H_{\hat{\mathbf{s}}}$, and $\mathcal{S}$ knows $\mathbf{e}_{k,j}[l]$ and $\Delta_j[l]$ for each $l \in S_j$. Then, $\mathcal{S}$ computes $\mathsf{M}_j[x_{k,j}] := \mathsf{M}'_j[x_{k,j}] + \mathbf{e}_{k,j} * \Delta_j$ for $k \in [\ell']$.
8. If there exists two different $j, j' \notin A$ such that $x_{k,j} \neq x_{k,j'}$ for some $k \in [\ell]$, then $\mathcal{S}$ aborts. Otherwise, for each $k \in [\ell]$, $\mathcal{S}$ sets $x_k := x_{k,j}$ for some $j \notin A$.
9. $\mathcal{S}$ sends $x_1, \ldots, x_\ell$ and $\mathsf{M}_j[x_1] := \mathsf{M}_j[x_{1,j}], \ldots, \mathsf{M}_j[x_\ell] := \mathsf{M}_j[x_{\ell,j}]$ to $\mathcal{F}_{\mathsf{aBit}}$.

This concludes the description of the simulation. Below, we show that the simulation is indistinguishable from the real protocol execution for two cases.

Analysis for Case 1. It is easy to see that the correlation and consistency checks pass in the case of honest party P_i. For $j \notin A \backslash \{i\}$, $\mathsf{M}_j[\mathbf{y}^i]$ sampled by $\mathcal{S}$ has the same distribution as the one in the real protocol execution, as $\mathsf{K}_j[\mathbf{y}^i] = \sum_{k=1}^{\ell'} \chi_k \cdot \mathsf{K}_j[x_k]$ is uniformly random in $\mathbb{F}_{2^\kappa}$. Below, all we need to do is to prove that $\mathbf{y}^i$ and $\mathsf{M}_j[\mathbf{y}^i]$ for $j \in A$ sent by $\mathcal{S}$ are statistically indistinguishable from the values sent by P_i in the real protocol execution.

Recall that in the real protocol execution, honest party P_i sends the following value:

$$\mathbf{y}^i = \sum_{k=1}^{\ell'} \chi_k \cdot x_k = \sum_{k=1}^{\ell} \chi_k \cdot x_k + \sum_{k=\ell+1}^{\ell+\kappa+\rho} \chi_k \cdot x_k.$$

The second summation corresponds to the image of a linear map $\psi : \mathbb{F}_2^{\kappa+\rho} \mapsto \mathbb{F}_2^\kappa$. From Lemma 3, we know that the map ψ has full rank with probability $1 - 2^{-\rho}$. In this case, $\sum_{k=\ell+1}^{\ell+\kappa+\rho} \chi_k \cdot x_k$ is uniformly random in $\mathbb{F}_{2^\kappa}$, since $(x_{\ell+1}, \ldots, x_{\ell+\kappa+\rho})$ are sampled uniformly at random by honest P_i. Thus $\mathbf{y}^i$ in the real world is statistically indistinguishable from the value simulated by $\mathcal{S}$. Finally, $\mathsf{M}_j[\mathbf{y}^i]$ has the same distribution in both worlds, since there is only one $\mathsf{M}_j[\mathbf{y}^i]$ satisfying the equation $\mathsf{M}_j[\mathbf{y}^i] = \mathsf{K}_j[\mathbf{y}^i] + \mathbf{y}^i \cdot \Delta_j$.

Analysis for Case 2. Without loss of generality, we first fix an honest party $P_j \notin A$ and analyze the simulation of $\mathcal{S}$. In the real protocol execution, if P_j does not abort, then $\widehat{\mathsf{M}}_j[\mathbf{y}^i] = \mathsf{K}_j[\mathbf{y}^i] + \tilde{\mathbf{y}}^i \cdot \Delta_j = \mathsf{K}_j[\mathbf{y}^i] + \mathbf{y}^{i,j} \cdot \Delta_j + \mathbf{e}^{i,j} \cdot \Delta_j$, where $\mathsf{K}_j[\mathbf{y}^i] = \sum_{k=1}^{\ell'} \chi_k \cdot \mathsf{K}_j[x_{k,j}]$. Besides, we have that

$$\begin{aligned}
\mathsf{M}'_j[\mathbf{y}^i] &= \sum_{k=1}^{\ell'} \chi_k \cdot \mathsf{M}'_j[x_{k,j}] = \sum_{k=1}^{\ell'} \chi_k \cdot (\mathsf{K}_j[x_{k,j}] + x_{k,j} \cdot \Delta_j + \mathbf{e}_{k,j} * \Delta_j) \\
&= \sum_{k=1}^{\ell'} \chi_k \cdot \mathsf{K}_j[x_{k,j}] + \Big(\sum_{k=1}^{\ell'} \chi_k \cdot x_{k,j}\Big) \cdot \Delta_j + \sum_{k=1}^{\ell'} \chi_k \cdot (\mathbf{e}_{k,j} * \Delta_j) \\
&= \mathsf{K}_j[\mathbf{y}^i] + \mathbf{y}^{i,j} \cdot \Delta_j + \sum_{k=1}^{\ell'} \chi_k \cdot (\mathbf{e}_{k,j} * \Delta_j) \,.
\end{aligned}$$

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

Initialize: Upon receiving (init) from all parties, sample $\Delta_i \leftarrow \{0,1\}^\kappa$ for $i \notin A$ and receive $\Delta_i \in \{0,1\}^\kappa$ from the adversary for $i \in A$. Store Δ_i for $i \in [n]$ and send Δ_i to party P_i.

Authentication: Upon receiving (aShare, ℓ) from all parties, sample $x_1, \ldots, x_\ell \leftarrow \{0,1\}$ and generate authenticated shares $\{\langle x_k \rangle\}_{k\in[\ell]}$ by executing $\mathsf{AuthShare}(x_k)$ defined in Figure 1 for $k \in [\ell]$.

Selective failure leakage: Wait for the adversary to input $(\mathsf{leak}, i, S, \{\Delta'[k]\}_{k\in S})$. If P_i is honest, this functionality executes the macro $\mathsf{GKleak}(i, S, \{\Delta'[k]\}_{k\in S})$ defined in Figure 1.

Figure 12: Functionality for authenticated shares.

From $E_j = \widehat{\mathsf{M}}_j[\mathbf{y}^i] + \sum_{k=1}^{\ell'} \chi_k \cdot \mathsf{M}'_j[x_{k,j}] = \widehat{\mathsf{M}}_j[\mathbf{y}^i] + \mathsf{M}'_j[\mathbf{y}^i]$, we have $E_j = \sum_{k=1}^{\ell'} \chi_k \cdot \left(\mathbf{e}_{k,j} * \Delta_j\right) + \mathbf{e}^{i,j} \cdot \Delta_j$. Since $\mathcal{A}$ knows $\mathbf{e}_{k,j} * \Delta_j$ for $k \in [\ell']$ and E_j, we have that $\mathbf{e}^{i,j} = 0$ unless $\mathcal{A}$ learns the global key Δ_j. The probability that the P_j's check passes is $|S_{\Delta_j}| \cdot 2^{-\kappa} + 2^{-\kappa} = 2^{d_j - \kappa} + 2^{-\kappa}$. By Lemma 2, we have that S_{Δ_j} restricted to the bits corresponding to $H_{\hat{\mathsf{s}}}$ has entropy at least d_j. Therefore, with probability at most 2^{-d_j}, $\mathcal{A}$ guesses successfully the bits $\{\Delta_j[l]\}_{l\in H_{\hat{\mathsf{s}}}}$. Overall, the probability that $\mathbf{e}^{i,j} \neq 0$ and the P_j's check passes is $2^{-d_j} \cdot (2^{d_j-\kappa} + 2^{-\kappa}) = 2^{-\kappa} + 2^{-(\kappa+d_j)}$. Therefore, the probability, that $\mathcal{S}$ aborts in Step 4 of the simulation but the real protocol execution does not abort, is $2^{-\kappa} + 2^{-(\kappa+d_j)} = \mathsf{negl}(\kappa)$. As a result, if the protocol does not abort, we have that $E_j = \sum_{k=1}^{\ell'} \chi_k \cdot \left(\mathbf{e}_{k,j} * \Delta_j\right)$ in both worlds with probability $1 - \mathsf{negl}(\kappa)$.

If $S_j = \emptyset$, i.e., $\mathbf{e}_{k,j} = 0$ for all $k \in [\ell']$, then it is easy to see that the simulation of $\mathcal{S}$ is indistinguishable from the real protocol execution. Below, for each $j \notin A$, we only consider the case that $S_j \neq \emptyset$. If the equation $\mathbf{X}_j \cdot \mathbf{t}_j = E_j$ has no solutions, this means that the real protocol execution will abort, which is the same as the simulation. If this equation has a unique solution (i.e., $\mathbf{X}_j$ has rank $c_j = |S_j|$), then $\mathcal{S}$ can extract a guess made by $\mathcal{A}$ about global key Δ_j, and forwards a decision from $\mathcal{F}_{\mathsf{aBit}}$ to $\mathcal{A}$. Clearly, in this case, the simulation of $\mathcal{S}$ is indistinguishable from the real protocol execution. If this equation has at least two different solutions, it means that matrix $\mathbf{X}_j$ has rank $< c_j$. By Lemma 3, we know that this happens with probability at most 2^{-d_j}. In the real protocol execution, the probability that P_j does not abort is $|S_{\Delta_j}| \cdot 2^{-\kappa} + 2^{-\kappa} = 2^{d_j-\kappa} + 2^{-\kappa}$. In all, the probability, that $\mathcal{S}$ aborts in Step 5 of the simulation but the real protocol execution will not abort, is bounded by $2^{-d_j} \cdot (2^{d_j-\kappa} + 2^{-\kappa}) = 2^{-\kappa} + 2^{-(\kappa+d_j)}$, which is negligible in κ. From Lemma 4, the probability that $\mathcal{S}$ aborts in Step 8 of the simulation is negligible in κ. Therefore, the simulation of $\mathcal{S}$ is indistinguishable from the real protocol execution.

In all, we have that the simulation is statistically indistinguishable from the real protocol execution, except with probability at most $1/2^\rho$ for Case 1 and $8/2^\kappa$ for Case 2. The outputs of honest parties are either independent from the adversary's view or always determined uniquely by their independent inputs and the outputs of corrupt parties. Therefore, we obtain that the joint distribution of the outputs of honest parties and adversary $\mathcal{A}$ in the real world execution is statistically indistinguishable from the joint distribution the outputs of honest parties and simulator $\mathcal{S}$ in the ideal world execution, which completes the proof. □

C Complexity and Security of Our Authenticated Share Protocol

We have already described the authenticated share protocol Π_{aShare} in Section 3.2. In the following, we analyze the rounds and communication complexity for protocol Π_{aShare}. Then, we prove that protocol Π_{aShare} securely realizes functionality $\mathcal{F}_{\mathsf{aShare}}$ shown in Figure 12 in the $(\mathcal{F}_{\mathsf{aBit}}, \mathcal{F}_{\mathsf{Com}})$-hybrid model.

C.1 Communication Complexity

Now, we analyze the rounds and communication cost of protocol Π_{aShare} (Figure 5) involving the cost of our authenticated bit protocol Π_{aBit}. Without considering the base OT protocol in the initialization phase, our protocol Π_{aShare} needs 4 rounds of communication. When the base OT protocol such as [PVW08, CO15, CSW20] is used, one extra round is required, as random zero shares can be computed in parallel with the base OT protocol. Note that all the calls of $\mathcal{F}_{\mathsf{aBit}}$ (related to the executions of Π_{aBit}) can be made in parallel. The communication cost per party is dominated by $(n-1)(\ell + 2\kappa + \rho)\kappa$ bits for generating authenticated shares. The consistency check needs only about $5n\kappa$ bits of extra communication, which is negligible for a moderate large ℓ.

C.2 Proof of Security

In the following, we provide the detailed security proof of our protocol Π_{aShare}. First we can see that Π_{aShare} is correct when all parties are honest, because

$$\begin{aligned}\sum_{j=1}^{n} \mathbf{z}_i^j &= \mathbf{z}_i^i + \sum_{j \neq i} \mathbf{z}_i^j = (\mathbf{y}^i + \mathbf{y}) \cdot \Delta_i + \sum_{j \neq i} \left(\mathsf{K}_i[\mathbf{y}^j] + \mathsf{M}_i[\mathbf{y}^j]\right) \\ &= (\mathbf{y}^i + \mathbf{y}) \cdot \Delta_i + \sum_{j \neq i} \mathbf{y}^j \cdot \Delta_i = \mathbf{y} \cdot \Delta_i + \mathbf{y} \cdot \Delta_i = 0.\end{aligned}$$

In the (init) command of $\mathcal{F}_{\mathsf{aBit}}$, a corrupt party P_i may deviate the protocol by providing inconsistent inputs Δ_i with two different honest parties. We define $\Delta_{i,j}$ to be actual inputs used by corrupt P_i, i.e., P_i sends $(\mathsf{init}, j, \Delta_{i,j})$ to $\mathcal{F}_{\mathsf{aBit}}$. Without loss of generality, we pick an honest party P_{j_0} and fix $\Delta_i = \Delta_{i,j_0}$. We define $R_{i,j} := \Delta_{i,j} + \Delta_i$ for $j \neq i$, and thus $R_{i,j_0} = 0$. Note that $R_{i,j}$ is fixed in the initialization phase. In the following lemma, we prove that a corrupt party P_i is impossible to provide inconsistent global keys $\Delta_{i,j}$ with different honest parties $P_j \notin A$.

Lemma 5. *If all honest parties do not abort in protocol Π_{aShare}, then for every corrupted party $P_i \in A$, all the global keys $\Delta_{i,j}$ are consistent with probability $1 - 1/2^{\kappa}$, i.e., $R_{i,j} = 0$ for each $j \notin A$.*

Proof. In Step 6 of protocol Π_{aShare}, if all corrupt parties are semi-honest, then every corrupt party P_i broadcasts $\tilde{\mathbf{y}}^i$ and computes $\mathbf{y} := \sum_{i=1}^{n} \tilde{\mathbf{y}}^i$. However, every malicious party $P_i \in A$ may broadcast an adversarial value $\hat{\mathbf{y}}^i$, such that $\hat{\mathbf{y}} := \sum_{i \in A} \hat{\mathbf{y}}^i + \sum_{i \notin A} \tilde{\mathbf{y}}^i = \mathbf{y} + \mathbf{e}$, where $\mathbf{e}$ is an additive error of the adversary's choice. We define $\mathbf{z}_i^j$ to be the value committed by a party P_j when P_j behaves honestly. The corrupt parties may deviate the protocol by committing the values $\hat{\mathbf{z}}_i^k$ for $k \in A$, in such a way that $\sum_{k \in A} \hat{\mathbf{z}}_i^k = \sum_{k \in A} \mathbf{z}_i^k + E_i$, where E_i is an adversarially chosen error.

If a malicious party P_i tries to cheat, then it has to pass the check in Step 8 of protocol Π_{aShare}. Therefore, we have the following:

$$\begin{aligned}0 &= \textstyle\sum_{j \notin A} \mathbf{z}_i^j + \sum_{j \in A} \hat{\mathbf{z}}_i^j = \mathbf{z}_i^i + \sum_{j \neq i} \mathbf{z}_i^j + E_i \\ &= \left(\textstyle\sum_{j \neq i} \mathsf{K}_i[\mathbf{y}^j] + (\mathbf{y}^i + \mathbf{y} + \mathbf{e}) \cdot \Delta_i\right) + \textstyle\sum_{j \neq i} \mathsf{M}_i[\mathbf{y}^j] + E_i \\ &= \textstyle\sum_{j \neq i} \left(\mathsf{K}_i[\mathbf{y}^j] + \mathsf{M}_i[\mathbf{y}^j]\right) + (\mathbf{y}^i + \mathbf{y} + \mathbf{e}) \cdot \Delta_i + E_i \\ &= \textstyle\sum_{j \neq i} \mathbf{y}^j \cdot \Delta_{i,j} + (\mathbf{y}^i + \mathbf{y} + \mathbf{e}) \cdot \Delta_i + E_i \\ &= \textstyle\sum_{j \neq i} \mathbf{y}^j \cdot R_{i,j} + \left(\mathbf{y}^i + \mathbf{y} + \mathbf{e} + \sum_{j \neq i} \mathbf{y}^j\right) \cdot \Delta_i + E_i \\ &= \textstyle\sum_{j \neq i} \mathbf{y}^j \cdot R_{i,j} + \mathbf{e} \cdot \Delta_i + E_i.\end{aligned}$$

If a malicious party P_i provides inconsistent global keys, then there exists $j_0, j_1 \notin A$ such that $R_{i,j_0} \neq R_{i,j_1}$ and $j_0 \neq j_1$. Therefore, the attack requires the adversary to set $E_i + \mathbf{e} \cdot \Delta_i = \mathbf{y}^{j_0} \cdot R_{i,j_0} + \mathbf{y}^{j_1} \cdot R_{i,j_1}$. Due to the re-randomization by random zero-sharing, from the adversary's view, $\mathbf{y}^{j_0}$ and $\mathbf{y}^{j_1}$ are uniformly random additive shares of $\mathbf{y}$. Thus, the adversary succeeds to cheat with probability $2^{-\kappa}$. □

Based on Lemma 5, we easily prove the following theorem.

Theorem 5 (Theorem 2, restated)**.** *Protocol* Π_{aShare} *shown in Figure 5 securely realizes functionality* $\mathcal{F}_{\mathsf{aShare}}$ *in the* $(\mathcal{F}_{\mathsf{aBit}}, \mathcal{F}_{\mathsf{Com}})$*-hybrid model.*

Proof. It is easy to construct a simulator $\mathcal{S}$, since all parties only communicate to each other in the phase of consistency check and $\mathcal{S}$ is allowed to know the shares $r_1^i, \ldots, r_\kappa^i$ for each $i \notin A$. Specifically, for any PPT adversary $\mathcal{A}$, we construct a PPT simulator $\mathcal{S}$ with access to functionality $\mathcal{F}_{\mathsf{aShare}}$ as follows:

1. In the initialization phase, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{aBit}}$, and receives $(j, \Delta_{i,j})$ for $i \in A$ and $j \neq i$ from $\mathcal{A}$. On behalf of every $P_i \in A$, $\mathcal{S}$ defines and sends $\Delta_i := \Delta_{i,j_0}$ for some $j_0 \notin A$ to $\mathcal{F}_{\mathsf{aShare}}$.

2. In the generation phase of authenticated shares, $\mathcal{S}$ plays the role of $\mathcal{F}_{\mathsf{aBit}}$ and records all the values received from $\mathcal{A}$. On behalf of every corrupt party P_i, $\mathcal{S}$ sends the corresponding shares, MACs and local keys to $\mathcal{F}_{\mathsf{aShare}}$. For each $i \notin A$, $\mathcal{S}$ also samples $r_1^i, \ldots, r_\kappa^i \leftarrow \{0,1\}$, and for each $h \in [\kappa]$, computes $\mathsf{M}_j[r_h^i]$ using the keys $\mathsf{K}_j[r_h^i]$ and $\Delta_{j,i}$ from $\mathcal{A}$ if $j \in A$ and samples $\mathsf{M}_j[r_h^i] \leftarrow \{0,1\}^\kappa$ otherwise.

3. When $\mathcal{S}$ plays the role of $\mathcal{F}_{\mathsf{aBit}}$, upon receiving the (leak) queries from $\mathcal{A}$, $\mathcal{S}$ forwards these queries to $\mathcal{F}_{\mathsf{aShare}}$, and sends the decision results from $\mathcal{F}_{\mathsf{aShare}}$ to $\mathcal{A}$. If $\mathcal{F}_{\mathsf{aShare}}$ aborts, $\mathcal{S}$ aborts. Otherwise, $\mathcal{S}$ continues to the simulation.

4. For each $i \notin A$, $\mathcal{S}$ samples a dummy global key $\Delta_i \leftarrow \{0,1\}^\kappa$ such that Δ_i is consistent with the real global key of P_i on the bits that have been leaked. For each $i \notin A$ and $h \in [\kappa]$, $\mathcal{S}$ defines $\mathsf{K}_i[r_h^j]$ using the corresponding $\mathsf{M}_i[r_h^j]$ and Δ_i.

5. $\mathcal{S}$ uses the values obtained in previous steps to perform the consistency check honestly on behalf of all honest parties. If the check fails, then $\mathcal{S}$ aborts.

6. If there are two different honest parties $j_0, j_1 \notin A$ such that $\Delta_{i,j_0} \neq \Delta_{i,j_1}$ for some $i \in A$, then $\mathcal{S}$ aborts.

In the above simulation, before $\mathcal{S}$ would abort, it outputs whatever $\mathcal{A}$ outputs. By Lemma 5, we guarantee the probability that $\mathcal{S}$ aborts in Step 5 is bounded by $2^{-\kappa}$. Therefore, it is easy to see that the simulation of $\mathcal{S}$ is statistically indistinguishable from the real protocol execution. Note that $\mathcal{S}$ does not know the real global keys of honest parties in the ideal world. Simulator $\mathcal{S}$ samples a dummy global key for every honest party to just perform the consistency check, and never uses these keys in any other place. Again, the outputs of honest parties are either independent from the adversary's view or always determined uniquely by their independent inputs and the outputs of corrupt parties. Therefore, we have that the joint distribution of the outputs of honest parties and adversary $\mathcal{A}$ in the real world execution is indistinguishable from the joint distribution the outputs of honest parties and simulator $\mathcal{S}$ in the ideal world execution, except with probability $2^{-\kappa}$. □

D Improved Authenticated Triple

In Appendix D.1, we present an optimized protocol for authenticated AND triples with the leakage of partial shares. Then, we show that the bucketing technique [NNOB12] for eliminating the leakage can still be applied in our setting in Appendix D.2.

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

Initialize: Upon receiving (init) from all parties, sample $\Delta_i \leftarrow \{0,1\}^\kappa$ for $i \notin A$ and receive $\Delta_i \in \{0,1\}^\kappa$ from the adversary for $i \in A$. Store Δ_i for $i \in [n]$ and send Δ_i to party P_i.

Triples: Upon receiving (LaAND) from all parties, sample $x, y \leftarrow \{0,1\}$, compute $z := x \wedge y$, and generate a random authenticated triple $(\langle x \rangle, \langle y \rangle, \langle z \rangle)$ by executing $\mathsf{AuthShare}(u)$ for each $u \in \{x, y, z\}$.

Selective failure queries for shares: Wait for the adversary to input $(Q, q, \{R_i\}_{i \notin A})$ where $Q \in \{0,1\}^\kappa$, $q \in \{0,1\}$ and $R_i \in \{0,1\}^\kappa$. This functionality checks that

$$Q \oplus \left(\bigoplus_{i \notin A} x^i R_i\right) = \left(q \oplus \bigoplus_{i \notin A} x^i \mathsf{lsb}(R_i)\right) \left(\bigoplus_{i \notin A} \Delta_i\right).$$

If the check fails, this functionality sends `fail` to all parties and aborts. Otherwise, this functionality sends `success` to the adversary, and proceeds as if nothing has happened.

Selective failure queries for global keys: Wait for the adversary to input $(\mathsf{leak}, i, S, \{\Delta'[k]\}_{k \in S})$. If P_i is honest, this functionality executes $\mathsf{GKleak}(i, S, \{\Delta'[k]\}_{k \in S})$ as defined in Figure 1.

Figure 13: Functionality for leaky AND triples.

D.1 Protocol for Leaky AND Triples

We first describe a functionality $\mathcal{F}_{\mathsf{LaAND}}$ for leaky authenticated AND triples in Figure 13. Then, we present an efficient protocol Π_{LaAND} shown in Figure 14 that securely computes $\mathcal{F}_{\mathsf{LaAND}}$ in the $(\mathcal{F}_{\mathsf{aShare}}, \mathcal{F}_{\mathsf{Com}})$-hybrid model, where $\mathsf{H} : \{0,1\}^{2\kappa} \to \{0,1\}^\kappa$ is a random oracle.

For functionality $\mathcal{F}_{\mathsf{LaAND}}$, similar to prior works, an adversary $\mathcal{A}$ is allowed to guess a share $x^{i^*} \in \{0,1\}$ of an honest party P_{i^*}. An incorrect guess will be caught immediately, while a correct guess keep undetected. In more detail, $\mathcal{A}$ does not directly learn the share x^{i^*}, but instead is allowed to make a query on some linear combination of x^{i^*} and Δ_{i^*}. In this special way, $\mathcal{A}$ cannot obtain more information than making a query on x^{i^*} and Δ_{i^*} directly. Moreover, $\mathcal{A}$ cannot learn any information on y^{i^*} or z^{i^*}.

For the protocol Π_{LaAND} shown in Figure 14, we require that $\mathcal{F}_{\mathsf{aShare}}$ generates global keys $\{\Delta_i\}_{i \in [n]}$ such that $\bigoplus_{i \in [n]} \mathsf{lsb}(\Delta_i) = 1$, e.g., $\mathsf{lsb}(\Delta_i) = 1$ if $i \neq 1$ and $\mathsf{lsb}(\Delta_i) = n \mod 2$ otherwise. In protocol Π_{LaAND}, we add a tweak $i \| j \| t$ to the computation of hash function H for generating the t-th leaky AND triple. It aims to prevent the attack described in [GKWY19] that a malicious party P_j may send the same share and MAC in multiple executions. In addition, we do not let a party P_i straightforwardly broadcast a bit d_i. Instead, we make the party commit to d_i, and then open it. This because the simulator needs to know the bits from the adversary before sending a bit d_i on behalf of honest party P_i, in the security proof.

For the sake of simplicity, we only describe one leaky AND triple generation in protocol Π_{LaAND}. When ℓ leaky authenticated AND triples need to be computed, we can run ℓ executions of protocol Π_{LaAND} in parallel with the same initialization, where all parties send $(\mathsf{aShare}, 3\ell)$ to $\mathcal{F}_{\mathsf{aShare}}$. In this case, we can further reduce the communication complexity by combining ℓ commitments into one commitment in a natural way.

Optimization and communication complexity. When the parties need to check that $\bigoplus_{i \in [n]} T_{i,t} = 0$ for $t \in [\ell]$, every party must open ℓ values in protocol Π_{LaAND}, which leads to $\ell\kappa$ bits of communication. We can reduce the communication to only κ bits by using the following batched check procedure.

1. After all $T_{i,t}$ for $i \in [n], t \in [\ell]$ have been computed, the parties call $\mathcal{F}_{\mathsf{Rand}}$ to generate random coefficients $\chi_1, \ldots, \chi_\ell \in \mathbb{F}_{2^\kappa}$.

2. Every party P_i computes $V_i := \sum_{t=1}^{\ell} \chi_t \cdot T_{i,t}$ (with arithmetic over $\mathbb{F}_{2^\kappa}$) and commits to V_i via calling functionality $\mathcal{F}_{\mathsf{Com}}$.

Protocol Π_{LaAND}

Initialize: All parties send (init) to $\mathcal{F}_{\mathsf{aShare}}$, which returns $\Delta_i \in \{0,1\}^\kappa$ to P_i for $i \in [n]$ such that $\bigoplus_{i\in[n]} \mathsf{lsb}(\Delta_i) = 1$.

Generate leaky AND triples: The parties generate the t-th leaky AND triple as follows:

1. All parties send $(\mathsf{aShare}, 3)$ to $\mathcal{F}_{\mathsf{aShare}}$, which returns random authenticated shares $\langle x\rangle, \langle y\rangle, \langle r\rangle$ to the parties.

   If receiving `fail` from functionality $\mathcal{F}_{\mathsf{aShare}}$, the parties abort.

2. For each $i \in [n]$, P_i locally computes $\Phi_i := y^i\Delta_i \oplus \left(\bigoplus_{k\neq i}(\mathsf{K}_i[y^k] \oplus \mathsf{M}_k[y^i])\right)$.

3. For each ordered pair (P_i, P_j) where $i \neq j$, P_i computes

$$\mathsf{K}_i[x^j]_{\Phi_i} := \mathsf{H}(\mathsf{K}_i[x^j], i\|j\|t) \text{ and } U_{i,j} := \mathsf{K}_i[x^j]_{\Phi_i} \oplus \mathsf{H}(\mathsf{K}_i[x^j] \oplus \Delta_i, i\|j\|t) \oplus \Phi_i,$$

   and then sends $U_{i,j}$ to P_j. Upon receiving $U_{i,j}$ from P_i, P_j computes

$$\mathsf{M}_i[x^j]_{\Phi_i} := x^j \cdot U_{i,j} \oplus \mathsf{H}(\mathsf{M}_i[x^j], i\|j\|t).$$

4. For each $i \in [n]$, P_i executes as follows:

   (a) Compute the following value

$$S_i := x^i\Phi_i \oplus \left(\bigoplus_{k\neq i}(\mathsf{K}_i[x^k]_{\Phi_i} \oplus \mathsf{M}_k[x^i]_{\Phi_k})\right) \oplus r^i\Delta_i \oplus \left(\bigoplus_{k\neq i}(\mathsf{K}_i[r^k] \oplus \mathsf{M}_k[r^i])\right).$$

   (b) Commit to $d_i := \mathsf{lsb}(S_i)$ by calling the (Commit) command of $\mathcal{F}_{\mathsf{Com}}$.

   (c) After all commitments have been made, open its commitment via calling the (Open) command of $\mathcal{F}_{\mathsf{Com}}$, and then compute $d := \bigoplus_{i\in[n]} d_i$.

5. For each $i \in [n]$, P_i computes and commits to $T_i := S_i \oplus d\Delta_i$ by calling the (Commit) command of $\mathcal{F}_{\mathsf{Com}}$.

6. For each $i \in [n]$, after all commitments have been made, all parties open their commitments by calling the (Open) command of $\mathcal{F}_{\mathsf{Com}}$, and then check that $\bigoplus_{i\in[n]} T_i = 0$. If the check fails, the parties abort.

7. For each $i \neq 1$, the parties define $[z^i]_i := [r^i]_i$. The parties also compute $[z^1]_1 := [r^1]_1 \oplus d$.

8. The parties output a leaky AND triple $(\langle x\rangle, \langle y\rangle, \langle z\rangle)$.

Figure 14: **Protocol for leaky authenticated AND triples in the $(\mathcal{F}_{\mathsf{aShare}}, \mathcal{F}_{\mathsf{Com}})$-hybrid model.** For a bit x and $i \in [n]$, $\mathsf{K}_i[x]_{\Phi_i}$ and $\mathsf{M}_i[x]_{\Phi_i} = \mathsf{K}_i[x]_{\Phi_i} \oplus x\Phi_i$ denote the local key and MAC respectively associated with a global key Φ_i.

3. After all commitments have been made, all parties open their commitments by calling $\mathcal{F}_{\mathsf{Com}}$ and check that $\bigoplus_{i\in[n]} V_i = 0$.

All the coefficients $\{\chi_t\}_{t\in[\ell]}$ are uniformly random after the values $\{T_{i,t}\}_{i\in[n],t\in[\ell]}$ have been defined. Therefore, $\bigoplus_{i\in[n]} V_i = 0$ implies that $\bigoplus_{i\in[n]} T_{i,t} = 0$ for $t \in [\ell]$ except with probability $2^{-\kappa}$. For selective failure queries of shares, this functionality can check a random linear combination of errors chosen by the adversary. As such, the adversary can still guess a bit x^i of honest party correctly with probability $1/2$, and an incorrect guess will be caught except with probability $1/2^\kappa$. The communication used to implement $\mathcal{F}_{\mathsf{Rand}}$ can be eliminated by using the Fiat-Shamir heuristic. In particular, the parties can compute the random coefficients by hashing the transcript, which is secure in the random oracle model.

With the above optimization, we analyze the communication rounds and complexity of Π_{LaAND} (Figure 14) in the $\mathcal{F}_{\mathsf{aShare}}$-hybrid model. When generating ℓ leaky AND triples, this protocol needs 5 rounds and about $\ell(\kappa+1)(n-1)$ bits of communication per party.

Proof of security for protocol Π_{LaAND}. To prepare for the security proof of our main protocol, we first show that: 1) our protocol is correct if all parties are honest; and 2) if the protocol execution does not abort, then the parties generate a correct authenticated AND triple with probability $1-\mathsf{negl}(\kappa)$.

Lemma 6. *Protocol Π_{LaAND} shown in Figure 14 would output a correct AND triple, if all parties are honest.*

Proof. According to the definition of Φ_i, we have the following:

$$\begin{aligned}
\bigoplus\nolimits_{i\in[n]} \Phi_i &= \bigoplus\nolimits_{i\in[n]} \left(y^i\Delta_i \oplus \bigoplus\nolimits_{k\neq i}\left(\mathsf{K}_i[y^k]\oplus \mathsf{M}_k[y^i]\right)\right)\\
&= \bigoplus\nolimits_{i\in[n]} \left(y^i\Delta_i \oplus \bigoplus\nolimits_{k\neq i}\left(\mathsf{K}_i[y^k]\oplus \mathsf{M}_i[y^k]\right)\right)\\
&= \bigoplus\nolimits_{i\in[n]} \left(y^i\Delta_i \oplus \bigoplus\nolimits_{k\neq i} y^k\Delta_i\right)\\
&= \left(\bigoplus\nolimits_{i\in[n]} y^i\right)\left(\bigoplus\nolimits_{i\in[n]} \Delta_i\right).
\end{aligned}$$

Note that $\mathsf{K}_i[x^j]_{\Phi_i} \oplus \mathsf{M}_i[x^j]_{\Phi_i}$ is equal to:

$$\begin{aligned}
&= \mathsf{H}(\mathsf{K}_i[x^j], i\|j\|t) \oplus \mathsf{H}(\mathsf{M}_i[x^j], i\|j\|t) \oplus x^j \cdot U_{i,j}\\
&= \mathsf{H}(\mathsf{K}_i[x^j], i\|j\|t) \oplus \mathsf{H}(\mathsf{K}_i[x^j]\oplus x^j\Delta_i, i\|j\|t) \oplus\\
&\quad x^j\cdot\left(\mathsf{H}(\mathsf{K}_i[x^j], i\|j\|t) \oplus \mathsf{H}(\mathsf{K}_i[x^j]\oplus \Delta_i, i\|j\|t)\oplus\Phi_i\right)\\
&= \mathsf{H}(\mathsf{K}_i[x^j], i\|j\|t) \oplus \mathsf{H}(\mathsf{K}_i[x^j], i\|j\|t) \oplus x^j\cdot\Phi_i = x^j\Phi_i.
\end{aligned}$$

Taking the above two equations, we have that

$$\begin{aligned}
\bigoplus\nolimits_{i\in[n]} S_i &= \bigoplus\nolimits_{i\in[n]}\left(x^i\Phi_i \oplus \bigoplus\nolimits_{k\neq i}\left(\mathsf{K}_i[x^k]_{\Phi_i}\oplus \mathsf{M}_k[x^i]_{\Phi_k}\right)\oplus r^i\Delta_i \oplus \bigoplus\nolimits_{k\neq i}\left(\mathsf{K}_i[r^k]\oplus \mathsf{M}_k[r^i]\right)\right)\\
&= \bigoplus\nolimits_{i\in[n]}\left(x^i\Phi_i \oplus \bigoplus\nolimits_{k\neq i}\left(\mathsf{K}_i[x^k]_{\Phi_i}\oplus \mathsf{M}_i[x^k]_{\Phi_i}\right)\right)\oplus \bigoplus\nolimits_{i\in[n]}\left(r^i\Delta_i \oplus \bigoplus\nolimits_{k\neq i}\left(\mathsf{K}_i[r^k]\oplus \mathsf{M}_i[r^k]\right)\right)\\
&= \bigoplus\nolimits_{i\in[n]}\left(x^i\Phi_i \oplus \bigoplus\nolimits_{k\neq i} x^k\Phi_i\right)\oplus \bigoplus\nolimits_{i\in[n]}\left(r^i\Delta_i \oplus \bigoplus\nolimits_{k\neq i} r^k\Delta_i\right)\\
&= \left(\bigoplus\nolimits_{i\in[n]} x^i\right)\left(\bigoplus\nolimits_{i\in[n]} \Phi_i\right)\oplus\left(\bigoplus\nolimits_{i\in[n]} r^i\right)\left(\bigoplus\nolimits_{i\in[n]} \Delta_i\right)\\
&= \left(\left(\bigoplus\nolimits_{i\in[n]} x^i\right)\wedge\left(\bigoplus\nolimits_{i\in[n]} y^i\right)\oplus\left(\bigoplus\nolimits_{i\in[n]} r^i\right)\right)\left(\bigoplus\nolimits_{i\in[n]} \Delta_i\right).
\end{aligned}$$

Since $\mathsf{lsb}(\bigoplus_{i\in[n]}\Delta_i) = 1$, it holds that

$$d = \mathsf{lsb}(\bigoplus\nolimits_{i\in[n]} S_i) = \left(\bigoplus\nolimits_{i\in[n]} x^i\right)\wedge\left(\bigoplus\nolimits_{i\in[n]} y^i\right)\oplus\left(\bigoplus\nolimits_{i\in[n]} r^i\right).$$

From $z^i = r^i$ and $z^1 = r^1 \oplus d$, we have $\left(\bigoplus_{i\in[n]} x^i\right)\wedge\left(\bigoplus_{i\in[n]} y^i\right) = d\oplus\left(\bigoplus_{i\in[n]} r^i\right) = \bigoplus_{i\in[n]} z^i$. Finally, it is easy to see that the following holds: $\bigoplus_{i\in[n]} T_i = \bigoplus_{i\in[n]}(S_i\oplus d\Delta_i) = \left(\bigoplus_{i\in[n]} S_i\right)\oplus d\left(\bigoplus_{i\in[n]}\Delta_i\right) = 0$. Therefore, no parties would abort. □

Lemma 7. *Let A be the set of malicious parties. If the honest parties do not abort, then the parties would output a correct AND triple such that*

$$\left(\bigoplus\nolimits_{i\in[n]} x^i\right)\wedge\left(\bigoplus\nolimits_{i\in[n]} y^i\right) = \bigoplus\nolimits_{i\in[n]} z^i,$$

where $\left\{z^i := r^i\right\}_{i\neq 1}$ and $z^1 := r^1 \oplus d'$, the bit d' is computed in Step 4c of protocol Π_{LaAND}, and $\left\{x^i, y^i, r^i\right\}_{i=1}^n$ are defined from authenticated shares $\langle x\rangle, \langle y\rangle, \langle r\rangle$ output by $\mathcal{F}_{\mathsf{aShare}}$.

Proof. Let $U'_{i,j}, d', S'_i, T'_i$ denote the values computed by a party P_i in the protocol Π_{LaAND} when some malicious parties deviate the protocol, and $U_{i,j}, d, S_i, T_i$ be the values that P_i would have computed when all parties are honest. For each $i \in A$, we define $R_{i,j} := U'_{i,j} \oplus U_{i,j}$ for each $j \notin A$ and $Q_i := T'_i \oplus T_i$. For each $j \notin A$, we also define $R_j := \bigoplus_{k \in A} R_{k,j}$.

Firstly, we show that if $d' = d = \left(\bigoplus_{i\in[n]} x^i\right) \wedge \left(\bigoplus_{i\in[n]} y^i\right) \oplus \left(\bigoplus_{i\in[n]} r^i\right)$, $\bigoplus_{i\in[n]} z^i = \left(\bigoplus_{i\in[n]} x^i\right) \wedge \left(\bigoplus_{i\in[n]} y^i\right)$ holds with probability 1. Since $z^i = r^i$ for $i \neq 1$ and $z^1 = r^1 \oplus d'$, we have:

$$\bigoplus_{i\in[n]} z^i = \left(\bigoplus_{i\in[n]} r^i\right) \oplus d' = \left(\bigoplus_{i\in[n]} x^i\right) \wedge \left(\bigoplus_{i\in[n]} y^i\right).$$

Below, we assume that $d' \neq d$ while at the same time that the check passes, and we will derive a contradiction from this. For each $i \in A$, an honest party $P_j \notin A$ would compute $\mathsf{M}'_i[x^j]_{\Phi_i} := x^j \cdot U'_{i,j} \oplus \mathsf{H}(\mathsf{M}_i[x^j], i\|j\|t) = x^j \cdot U_{i,j} \oplus \mathsf{H}(\mathsf{M}_i[x^j], i\|j\|t) \oplus x^j \cdot R_{i,j} = \mathsf{M}_i[x^j]_{\Phi_i} \oplus x^j \cdot R_{i,j}$. Then P_j will compute $S'_j = S_j \oplus \left(\bigoplus_{k\in A} x^j R_{k,j}\right) = S_j \oplus x^j \cdot R_j$. Note that we have $\bigoplus_{i\in[n]} T_i = 0$. Thus, we know that

$$\begin{aligned}
\bigoplus_{i\in[n]} T'_i &= \bigoplus_{i\in A} T'_i \oplus \bigoplus_{i\notin A} T'_i \\
&= \bigoplus_{i\in A} (T_i \oplus Q_i) \oplus \bigoplus_{i\notin A} (S'_i \oplus d'\Delta_i) \\
&= \bigoplus_{i\in A} (T_i \oplus Q_i) \oplus \bigoplus_{i\notin A} \left(S_i \oplus x^i R_i \oplus d\Delta_i \oplus \Delta_i\right) \\
&= \bigoplus_{i\in[n]} T_i \oplus \bigoplus_{i\in A} Q_i \oplus \bigoplus_{i\notin A} x^i R_i \oplus \bigoplus_{i\notin A} \Delta_i \\
&= \bigoplus_{i\in A} Q_i \oplus \bigoplus_{i\notin A} x^i R_i \oplus \bigoplus_{i\notin A} \Delta_i.
\end{aligned}$$

To make $\bigoplus_{i\in[n]} T'_i = 0$, the adversary must find errors such that

$$\bigoplus_{i\in A} Q_i \oplus \bigoplus_{i\notin A} x^i R_i = \bigoplus_{i\notin A} \Delta_i \tag{1}$$

We here consider the case that there is only one honest party, because if there are at least two honest parties, adversary $\mathcal{A}$ will have a lower probability to guarantee the above equation (1) holds. Let $P_{i^*} \notin A$ be the unique honest party. If $\mathcal{A}$ succeeds to guess c bits of Δ_{i^*} for some $c \in [\kappa] \cup \{0\}$ via the (leak) command of $\mathcal{F}_{\mathsf{aShare}}$, the protocol will abort except with probability $1/2^c$. If $\mathcal{A}$ makes at most q queries to random oracle H, then it will learn Δ_{i^*} from $\{U_{i^*,j}\}_{j\neq i^*}$ sent by P_{i^*} with probability at most $q/2^{\kappa-1-c}$. Therefore, the probability, that the protocol does not abort and the above equation (1) holds, is bounded by $q/2^{\kappa-1-c} \cdot 1/2^c = q/2^{\kappa-1}$. □

Theorem 6. *Let* H *be a random oracle. Protocol* Π_{LaAND} *shown in Figure 14 securely realizes functionality* $\mathcal{F}_{\mathsf{LaAND}}$ *in the* $(\mathcal{F}_{\mathsf{aShare}}, \mathcal{F}_{\mathsf{Com}})$*-hybrid model.*

Proof. Let $\mathcal{A}$ be a PPT adversary who corrupts a subset of parties A. We construct a PPT simulator $\mathcal{S}$ with access to functionality $\mathcal{F}_{\mathsf{LaAND}}$, which runs $\mathcal{A}$ as a subroutine and simulates $\mathcal{A}$'s view. Before $\mathcal{S}$ aborts, it outputs whatever $\mathcal{A}$ outputs.

Description of the simulation.

1. When playing the role of $\mathcal{F}_{\mathsf{aShare}}$, $\mathcal{S}$ receives global key Δ_i and P_i's authenticated shares of $\langle x\rangle, \langle y\rangle, \langle r\rangle$ from $\mathcal{A}$ for each $i \in A$. Then $\mathcal{S}$ samples $d \leftarrow \{0,1\}$, and defines $[z^i]_i := [r^i]_i$ for each $i \neq 1$ and $[z^1]_1 := [r^1]_1 \oplus d$. For $i \in A$, $\mathcal{S}$ sends Δ_i and P_i's authenticated shares for $(\langle x\rangle, \langle y\rangle, \langle z\rangle)$ to $\mathcal{F}_{\mathsf{LaAND}}$.

2. For all (leak) queries on global keys of honest parties from $\mathcal{A}$ against $\mathcal{F}_{\mathsf{aShare}}$, $\mathcal{S}$ forwards these queries to $\mathcal{F}_{\mathsf{LaAND}}$, and then sends the decision results to $\mathcal{A}$. If $\mathcal{F}_{\mathsf{LaAND}}$ aborts, $\mathcal{S}$ aborts.

3. For each $i \notin A$, $\mathcal{S}$ picks a random $U_{i,j} \leftarrow \{0,1\}^\kappa$ as a message sent from P_i to P_j for each $j \neq i$. For each $i \in A$, using global key Δ_i and the P_i's authenticated shares of $\langle x \rangle, \langle y \rangle, \langle r \rangle$, $\mathcal{S}$ computes locally $U_{i,j}$ for each $j \notin A, j \neq i$, $d_i := \mathsf{lsb}(S_i)$ and $T_i := S_i \oplus d\Delta_i$, which will be sent by a semi-honest party P_i, where S_i is the value computed by semi-honest party P_i with its authenticated shares and the messages $\{U_{j,i}\}_{j \neq i}$.

4. For each $i \notin A$, $\mathcal{S}$ acts as honest party P_i and sends $U_{i,j}$ sampled in the previous step to P_j for each $j \neq i$. For each $i \in A$, for every $j \notin A, j \neq i$, $\mathcal{S}$ acts as honest party P_j and receives $U'_{i,j}$ from $\mathcal{A}$, and then computes $R_{i,j} := U'_{i,j} \oplus U_{i,j}$. For each $i \notin A$, $\mathcal{S}$ computes $R_i := \bigoplus_{k \in A} R_{k,i}$.

5. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{Com}}$ and receives d'_i for each $i \in A$ from $\mathcal{A}$. Then, $\mathcal{S}$ computes $q_i := d'_i \oplus d_i$ and $q := \bigoplus_{i \in A} q_i$. By Lemma 7, we know that $d' = \bigoplus_{i \in [n]} d'_i$ is equal to $d = \bigoplus_{i \in [n]} d_i$ in the real protocol execution with probability $1 - \mathsf{negl}(\kappa)$. Therefore, $\mathcal{S}$ sets $d' := d$. For each $i \notin A$, $\mathcal{S}$ samples $d'_i \leftarrow \{0,1\}$ such that $\bigoplus_{i \in [n]} d'_i = d' = d$. Then, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{Com}}$ and opens d'_i for each $i \notin A$ to all parties.

6. $\mathcal{S}$ plays the role of $\mathcal{F}_{\mathsf{Com}}$, and receives T'_i from every corrupt party $P_i \in A$. $\mathcal{S}$ computes $Q_i := T'_i \oplus T_i$ for each $i \in A$, and then computes $Q := \bigoplus_{i \in A} Q_i$. Then, $\mathcal{S}$ sends $(Q, q, \{R_i\}_{i \notin A})$ to $\mathcal{F}_{\mathsf{LaAND}}$ as a selective failure query on x-shares. If $\mathcal{F}_{\mathsf{LaAND}}$ aborts, $\mathcal{S}$ aborts. Otherwise, for each $i \notin A$, $\mathcal{S}$ picks $T'_i \leftarrow \{0,1\}^\kappa$ such that $\mathsf{lsb}(T'_i) = d'_i \oplus d' \cdot \mathsf{lsb}(\Delta_i)$ and $\bigoplus_{i \in [n]} T'_i = 0$, and then opens it to all parties.

For each $i \notin A$, we assume that $\mathcal{A}$ guesses c_i bits of Δ_i for some $c_i \in [\kappa] \cup \{0\}$ with probability of aborting $1 - 1/2^{c_i}$. Since H is a random oracle, the probability that $(\mathsf{M}_i[x^j] \oplus \Delta_i, i \| j \| t)$ for $j \neq i$ has been queried is bounded by $q/2^{\kappa-1-c_i}$, where q is the number of queries to H. Therefore, for each $i \notin A, j \neq i$, random value $U_{i,j}$ simulated by $\mathcal{S}$ is indistinguishable from the value in the real protocol execution, except with probability at most $1/2^{c_i} \cdot q/2^{\kappa-1-c_i} = q/2^{\kappa-1}$, which is negligible in κ. In the $\mathcal{F}_{\mathsf{aShare}}$-hybrid model, the shares of all honest parties for $\langle y \rangle, \langle r \rangle$ are uniform and kept secret from the adversary's view. Therefore, $\{d'_i\}_{i \notin A}$ simulated by $\mathcal{S}$ have the same distribution as the bits sent in the real protocol execution.

Below, we show that the probability of aborting due to the selective failure attack in the real world is the same as the one in the ideal world. By the proof of Lemma 7, we have that $S'_i = S_i \oplus x^i \cdot R_i$. Thus, for each $i \notin A$, $d'_i = d_i \oplus x^i \cdot \mathsf{lsb}(R_i)$. Due to $d'_i = d_i \oplus q_i$ for each $i \in A$, we know that

$$\begin{aligned} d' &= \bigoplus\nolimits_{i \in [n]} d'_i = \bigoplus\nolimits_{i \notin A} d'_i \oplus \bigoplus\nolimits_{i \in A} d'_i \\ &= \bigoplus\nolimits_{i \in [n]} d_i \oplus \bigoplus\nolimits_{i \notin A} x^i \cdot \mathsf{lsb}(R_i) \oplus \bigoplus\nolimits_{i \in A} q_i \\ &= d \oplus \bigoplus\nolimits_{i \notin A} x^i \cdot \mathsf{lsb}(R_i) \oplus q. \end{aligned}$$

Based on the above equation, we have that

$$\begin{aligned} \bigoplus_{i \in [n]} T'_i &= \bigoplus_{i \in A} T'_i \oplus \bigoplus_{i \notin A} T'_i \\ &= \bigoplus_{i \in A} (T_i \oplus Q_i) \oplus \bigoplus_{i \notin A} (S'_i \oplus d'\Delta_i) \\ &= \bigoplus_{i \in A} (T_i \oplus Q_i) \oplus \bigoplus_{i \notin A} \Big(T_i \oplus x^i R_i \oplus \big(\bigoplus_{j \notin A} x^j \mathsf{lsb}(R_j) \big) \Delta_i \oplus q\Delta_i \Big) \\ &= \bigoplus_{i \in [n]} T_i \oplus \bigoplus_{i \in A} Q_i \oplus \bigoplus_{i \notin A} x^i R_i \oplus \Big(q \oplus \bigoplus_{i \notin A} x^i \mathsf{lsb}(R_i) \Big) \Big(\bigoplus_{i \notin A} \Delta_i \Big) \\ &= Q \oplus \Big(\bigoplus_{i \notin A} x^i R_i \Big) \oplus \Big(q \oplus \bigoplus_{i \notin A} x^i \mathsf{lsb}(R_i) \Big) \Big(\bigoplus_{i \notin A} \Delta_i \Big). \end{aligned}$$

Therefore, $\bigoplus_{i\in[n]} T_i' = 0$ if and only if the following holds:

$$Q \oplus \left(\bigoplus_{i\notin A} x^i R_i\right) = \left(q \oplus \bigoplus_{i\notin A} x^i \mathsf{lsb}(R_i)\right)\left(\bigoplus_{i\notin A} \Delta_i\right),$$

which implies the same probability of aborting for both two worlds.

In the simulation of $\mathcal{S}$, if $\mathcal{F}_{\mathsf{LaAND}}$ does not abort, for each $i \notin A$, T_i' is chosen at random except for the least significant bit. We need to show that if the protocol does not abort, then $\{T_i'\}_{i\notin A}$ simulated by $\mathcal{S}$ is indistinguishable from the values opened in the real protocol execution. Firstly, we prove that $\bigoplus_{i\in[n]} \mathsf{lsb}(T_i') = 0$ with probability at least $1 - q/2^{\kappa-1}$, where q is an upper bound of the number of H queries. From a similar analysis of the proof of Lemma 7, we have that $Q = \bigoplus_{i\notin A} x^i R_i$ and $q = \bigoplus_{i\notin A} x^i \mathsf{lsb}(R_i)$, except with probability at most $q/2^{\kappa-1}$. Thus, with probability at least $1-q/2^{\kappa-1}$, $\mathcal{F}_{\mathsf{LaAND}}$ does not abort, $\mathsf{lsb}(Q) = q$ and $d' = d \oplus q \oplus \bigoplus_{i\notin A} x^i \mathsf{lsb}(R_i) = d$. From the simulation by simulator $\mathcal{S}$, we have that $\bigoplus_{i\in[n]} \mathsf{lsb}(T_i')$ is equal to:

$$\begin{aligned}
&= \bigoplus_{i\notin A} \left(d_i' \oplus d' \cdot \mathsf{lsb}(\Delta_i)\right) \oplus \bigoplus_{i\in A} \left(\mathsf{lsb}(T_i) \oplus \mathsf{lsb}(Q_i)\right) \\
&= \bigoplus_{i\notin A} d_i' \oplus d' \cdot \bigoplus_{i\notin A} \mathsf{lsb}(\Delta_i) \oplus \bigoplus_{i\in A} \left(d_i \oplus d \cdot \mathsf{lsb}(\Delta_i)\right) \oplus \bigoplus_{i\in A} \mathsf{lsb}(Q_i) \\
&= \bigoplus_{i\notin A} d_i' \oplus d' \cdot \bigoplus_{i\notin A} \mathsf{lsb}(\Delta_i) \oplus \bigoplus_{i\in A} (d_i' \oplus q_i) \oplus d \cdot \bigoplus_{i\in A} \mathsf{lsb}(\Delta_i) \oplus q \\
&= d' \oplus d \cdot \left(\bigoplus_{i\in[n]} \mathsf{lsb}(\Delta_i)\right) = d' \oplus d = 0.
\end{aligned}$$

Below, we prove if the protocol execution does not abort, then T_i' computed by honest party P_i is uniformly random under the condition that $\bigoplus_{i\in[n]} T_i' = 0$ and $\mathsf{lsb}(T_i') = d_i' \oplus d' \cdot \mathsf{lsb}(\Delta_i)$ in the real protocol execution. When only one party is honest, it is obvious that T_i' with $i \notin A$ is defined by the equation $\bigoplus_{i\in[n]} T_i' = 0$. In the following, we focus on the case that there are at least two honest parties. In particular, for each $i \notin A$, we define

$$F_i := \bigoplus_{k\neq i} \left(\mathsf{K}_i[r^k] \oplus \mathsf{M}_k[r^i]\right).$$

We show that for any proper subset $\mathsf{S} \subset [n]\backslash A$, $\bigoplus_{i\in\mathsf{S}} F_i$ is perfectly indistinguishable from a random value in $\{0,1\}^\kappa$. We use e to denote an honest party such that $e \notin A$ and $e \notin \mathsf{S}$. Such e always exists, as S is a proper subset of $[n]\backslash A$. We have the following holds:

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

From the above equation, we have that for $i \in \mathsf{S}$, $\mathsf{K}_e[r^i]$ is not in the computation, while $\mathsf{M}_e[r^i] = \mathsf{K}_e[r^i] \oplus r^i\Delta_e$ is. Since $\mathsf{K}_e[r^i]$ is uniform at random from $\mathcal{F}_{\mathsf{aShare}}$ and is kept unknown for $\mathcal{A}$ as both $i, e \notin A$, $\bigoplus_{i\in\mathsf{S}} F_i$ is random and unknown for $\mathcal{A}$. Therefore, for any proper subset $\mathsf{S} \subset [n]\backslash A$, $\bigoplus_{i\in\mathsf{S}} S_i'$ is

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

Initialize: Upon receiving (init) from all parties, sample $\Delta_i \leftarrow \{0,1\}^\kappa$ for $i \notin A$ and receive $\Delta_i \in \{0,1\}^\kappa$ from the adversary for $i \in A$. Store Δ_i for $i \in [n]$ and send Δ_i to party P_i.

Triples: Upon receiving (aAND, ℓ) from all parties, for each $k \in [\ell]$, sample $x_k, y_k \leftarrow \{0,1\}$, compute $z_k := x_k \wedge y_k$, and generate a random authenticated triple $(\langle x_k \rangle, \langle y_k \rangle, \langle z_k \rangle)$ by executing $\mathsf{AuthShare}(u_k)$ for each $u_k \in \{x_k, y_k, z_k\}$.

Selective failure leakage: Wait for the adversary to input $(\mathsf{leak}, i, S, \{\Delta'[k]\}_{k \in S})$. If P_i is honest, this functionality executes the macro $\mathsf{GKleak}(i, S, \{\Delta'[k]\}_{k \in S})$ as defined in Figure 1.

Figure 15: Functionality for authenticated AND triples.

Protocol Π_{aAND}

Initialize: All parties send (init) to $\mathcal{F}_{\mathsf{LaAND}}$, which returns $\Delta_i \in \{0,1\}^\kappa$ to P_i for $i \in [n]$.

Generate leaky AND triples: All parties set $\ell' := B \cdot \ell$ where B is the bucket size, and then call $\mathcal{F}_{\mathsf{LaAND}}$ ℓ' times and obtains ℓ' leaky authenticated AND triples $\{(\langle x_k \rangle, \langle y_k \rangle, \langle z_k \rangle)\}_{k \in [\ell']}$. If receiving `fail` from functionality $\mathcal{F}_{\mathsf{LaAND}}$, the parties abort.

Eliminate the leakage with bucketing: The parties eliminate the possible leakage on x-shares as follows.

1. All parties call $\mathcal{F}_{\mathsf{Rand}}$ to sample a random permutation π on $\{1, \ldots, \ell'\}$. Then the parties randomly partition all leaky AND triples into ℓ buckets of size B accordingly, i.e., for $j \in \{0, 1, \ldots, \ell - 1\}$, the B triples $\{(\langle x_{\pi(k)} \rangle, \langle y_{\pi(k)} \rangle, \langle z_{\pi(k)} \rangle)\}_{k=j \cdot B+1}^{j \cdot B+B}$ are defined to be in the j-th bucket.
2. For each bucket, the parties combine the B leaky AND triples into one non-leaky AND triple. We describe how to combine two leaky AND triples, calling them $(\langle x_1 \rangle, \langle y_1 \rangle, \langle z_1 \rangle)$ and $(\langle x_2 \rangle, \langle y_2 \rangle, \langle z_2 \rangle)$, into one calling the result $(\langle x \rangle, \langle y \rangle, \langle z \rangle)$. In particular, the parties execute as follows:
   (a) Compute $d := \mathsf{Open}(\langle y_1 \rangle \oplus \langle y_2 \rangle)$.
   (b) Set $\langle x \rangle := \langle x_1 \rangle \oplus \langle x_2 \rangle$, $\langle y \rangle := \langle y_1 \rangle$, and $\langle z \rangle := \langle z_1 \rangle \oplus \langle z_2 \rangle \oplus d \langle x_2 \rangle$.

   To combine all B leaky triples in the same bucket, the parties just iterate by taking the result and combine it with the next triple in the bucket.
3. All parties output the ℓ non-leaky AND triples.

Figure 16: Protocol for authenticated AND triples without leakage of shares.

indistinguishable from a random value, except that the least significant bit is revealed, where S'_i is the value computed by honest party P_i in Step 4a for $i \in \mathsf{S}$. Thus, for any proper subset $\mathsf{S} \subset [n] \backslash A$, $\bigoplus_{i \in \mathsf{S}} T'_i$ is indistinguishable from a random value except that $\mathsf{lsb}\left(\bigoplus_{i \in \mathsf{S}} T'_i\right)$ is fixed.

From Lemma 7, if $\left(\bigoplus_{i \in [n]} x^i\right) \wedge \left(\bigoplus_{i \in [n]} y^i\right) \neq \bigoplus_{i \in [n]} z^i$, then the real protocol execution will abort with probability $1 - q/2^{\kappa-1}$. Therefore, if honest parties do not abort, then protocol Π_{LaAND} will output a correct authenticated AND triple with probability $1 - \mathsf{negl}(\kappa)$, while functionality $\mathcal{F}_{\mathsf{LaAND}}$ always outputs a correct AND triple. In conclusion, we complete the proof. □

D.2 From Leaky Authenticated AND Triples to Authenticated AND Triples

Similar to prior works, we can eliminate the triple leakage based on bucketing. Based on the techniques in [NNOB12, WRK17b], we present an efficient protocol Π_{aAND} for authenticated AND triples, which securely computes a functionality $\mathcal{F}_{\mathsf{aAND}}$ shown in Figure 15. The details of Π_{aAND} are described in Figure 16. Our protocol is essentially the same as the one by Wang et al. [WRK17b], except that a) opening authenticated shares in an amortized way rather than directly sending the MACs; b) calling the functionality $\mathcal{F}_{\mathsf{LaAND}}$ for leaky AND triples with weak global keys. Given prior works [NNOB12, WRK17b], the security proof of protocol Π_{aAND} follows immediately, and thus is omitted. Note that although the adversary may leak a few bits of global keys via the selective failure attack, this has no impact on the security, by following the proof in Lemma 1.

According to Theorem 8 in [NNOB12], we have that $B = \frac{\rho}{\log \ell + 1} + 1$ such that the success probability of the adversary is bounded by $2^{-\rho}$. We analyze the communication rounds and complexity of protocol Π_{aAND} shown in Figure 16 in the $\mathcal{F}_{\mathsf{aShare}}$-hybrid model, including the cost of LaAND. Specifically, this protocol needs 5 rounds as the executions for $\mathcal{F}_{\mathsf{Rand}}$ and Open in Figure 16 can be merged with the final three rounds of Π_{LaAND}. Protocol Π_{aAND} needs to communicate about $B\ell(\kappa+1)(n-1) + (B-1)\ell(n-1)$ bits for each party per execution.

E Security Proof of Our MPC Protocol

In this section, we give a full proof of security to the protocol Π_{mpc} described in Section 4.

E.1 Related Lemmas

Prior to proceeding the main proof, we present four related lemmas. The first lemma addresses the correctness of our distributed garbling scheme in the honest case. The second lemma shows that malicious party P_1 can learn only one label generated by an honest party for each wire. The third lemma addresses the correctness of P_1's output when other parties are corrupted. The fourth lemma addresses the correctness of the output of honest party P_i with $i \neq 1$, when P_1 and other parties are corrupted. We omit the proof of correctness for generating authenticated shares of multiplication of two wire masks by using random authenticated AND triples (Step 7 of protocol Π_{mpc}), when some parties are corrupted. Recall that this procedure adopts a standard technique (i.e., authenticated Beaver triples [Bea92, BDOZ11]), and uses a random oracle H to perform the amortized opening of authenticated shares in which the security is proved in Appendix A.3.

Lemma 8. *When all parties follow the protocol description honestly, then after Step 10, for each wire w in the circuit, evaluator P_1 can obtain the correct public value Λ_w and garbled labels $\left\{\mathsf{L}^i_{w,\Lambda_w}\right\}_{i\neq 1}$.*

Proof. In the following, we prove this lemma by induction on the gates in the circuit.

Base step. It is easy to verify that this lemma holds for all circuit-input wires after input processing has been executed (Step 9).

Induction step. This lemma trivially holds for XOR gates. Thus, we focus on each AND gate $(\alpha, \beta, \gamma, \wedge)$. By the induction hypothesis, P_1 holds the correct $(\Lambda_\alpha, \{\mathsf{L}^i_{\alpha,\Lambda_\alpha}\}_{i\neq 1})$ and $(\Lambda_\beta, \{\mathsf{L}^i_{\beta,\Lambda_\beta}\}_{i\neq 1})$. Let $u = \Lambda_\alpha$ and $v = \Lambda_\beta$, P_1 evaluates the circuit as follows:

$$\left\{\mathsf{M}_j[r^1_{uv}] := \Lambda_\alpha \cdot \mathsf{M}_j[\lambda^1_\beta] \oplus \Lambda_\beta \cdot \mathsf{M}_j[\lambda^1_\alpha] \oplus \mathsf{M}_j[\lambda^1_{\alpha\beta}] \oplus \mathsf{M}_j[\lambda^1_\gamma]\right\}_{j\neq 1},$$

$$\left\{\mathsf{M}_j[r^i_{uv}] := \mathsf{H}(\mathsf{L}^i_{\alpha,\Lambda_\alpha}, \mathsf{L}^i_{\beta,\Lambda_\beta}, \gamma, j) \oplus G^{i,j}_{\gamma,uv}\right\}_{i\neq 1, j\neq i,1},$$

$$\left\{\mathsf{L}^i_{\gamma,\Lambda_\gamma} := \mathsf{H}(\mathsf{L}^i_{\alpha,\Lambda_\alpha}, \gamma) \oplus \mathsf{H}(\mathsf{L}^i_{\beta,\Lambda_\beta}, \gamma) \oplus \Lambda_\alpha \mathcal{G}^i_{\gamma,0} \oplus \Lambda_\beta(\mathcal{G}^i_{\gamma,1} \oplus \mathsf{L}^i_{\alpha,\Lambda_\alpha}) \oplus (\textstyle\bigoplus_{j\neq i} \mathsf{M}_i[r^j_{uv}])\right\}_{i\neq 1}.$$

Observe that for each $i \neq 1$, we have:

$$\begin{aligned}\mathsf{L}^i_{\gamma,\Lambda_\gamma} := {} & \Big(\mathsf{H}(\mathsf{L}^i_{\alpha,\Lambda_\alpha},\gamma) \oplus \Lambda_\alpha \mathcal{G}^i_{\gamma,0} \oplus \Lambda_\alpha(\textstyle\bigoplus_{j\neq i}\mathsf{M}_i[\lambda^j_\beta])\Big) \oplus \Big(\mathsf{H}(\mathsf{L}^i_{\beta,\Lambda_\beta},\gamma)\oplus \Lambda_\beta(\mathcal{G}^i_{\gamma,1}\oplus \mathsf{L}^i_{\alpha,\Lambda_\alpha})\oplus \\ & \Lambda_\beta(\textstyle\bigoplus_{j\neq i}\mathsf{M}_i[\lambda^j_\alpha])\Big) \oplus \Big(\textstyle\bigoplus_{j\neq i}\mathsf{M}_i[\lambda^j_{\alpha\beta}]\Big)\oplus\Big(\textstyle\bigoplus_{j\neq i}\mathsf{M}_i[\lambda^j_\gamma]\Big).\end{aligned}$$

It is easy to verify that the following holds:

$$\begin{aligned}& \mathsf{H}(\mathsf{L}^i_{\alpha,\Lambda_\alpha},\gamma) \oplus \Lambda_\alpha \mathcal{G}^i_{\gamma,0} \oplus \Lambda_\alpha(\textstyle\bigoplus_{j\neq i}\mathsf{M}_i[\lambda^j_\beta]) \\ & = \mathsf{H}(\mathsf{L}^i_{\alpha,\Lambda_\alpha},\gamma) \oplus \Lambda_\alpha\big(\mathsf{H}(\mathsf{L}^i_{\alpha,0},\gamma)\oplus\mathsf{H}(\mathsf{L}^i_{\alpha,1},\gamma)\big) \oplus \Lambda_\alpha(\textstyle\bigoplus_{j\neq i}\mathsf{K}_i[\lambda^j_\beta]\oplus\lambda^i_\beta\Delta_i\oplus\textstyle\bigoplus_{j\neq i}\mathsf{M}_i[\lambda^j_\beta]) \\ & = \mathsf{H}(\mathsf{L}^i_{\alpha,0},\gamma)\oplus\Lambda_\alpha\lambda_\beta\Delta_i \\ & \text{and} \\ & \mathsf{H}(\mathsf{L}^i_{\beta,\Lambda_\beta},\gamma)\oplus \Lambda_\beta(\mathcal{G}^i_{\gamma,1}\oplus \mathsf{L}^i_{\alpha,\Lambda_\alpha})\oplus\Lambda_\beta(\textstyle\bigoplus_{j\neq i}\mathsf{M}_i[\lambda^j_\alpha]) \\ & = \mathsf{H}(\mathsf{L}^i_{\beta,\Lambda_\beta},\gamma)\oplus\Lambda_\beta\big(\mathsf{H}(\mathsf{L}^i_{\beta,0},\gamma)\oplus\mathsf{H}(\mathsf{L}^i_{\beta,1},\gamma)\big)\oplus\Lambda_\beta(\mathsf{L}^i_{\alpha,0}\oplus\mathsf{L}^i_{\alpha,\Lambda_\alpha}) \\ & \quad \oplus\Lambda_\beta(\textstyle\bigoplus_{j\neq i}\mathsf{K}_i[\lambda^j_\alpha]\oplus\textstyle\bigoplus_{j\neq i}\mathsf{M}_i[\lambda^j_\alpha]\oplus\lambda^i_\alpha\Delta_i) \\ & = \mathsf{H}(\mathsf{L}^i_{\beta,0},\gamma)\oplus\Lambda_\alpha\Lambda_\beta\Delta_i\oplus\Lambda_\beta\lambda_\alpha\Delta_i.\end{aligned}$$

Each garbler P_i locally computes the 0-label $\mathsf{L}^i_{\gamma,0}$ as:

$$\mathsf{L}^i_{\gamma,0} := \mathsf{H}(\mathsf{L}^i_{\alpha,0},\gamma)\oplus\mathsf{H}(\mathsf{L}^i_{\beta,0},\gamma)\oplus\Big(\textstyle\bigoplus_{j\neq i}\mathsf{K}_i[\lambda^j_{\alpha\beta}]\Big)\oplus\lambda^i_{\alpha\beta}\Delta_i\oplus\Big(\textstyle\bigoplus_{j\neq i}\mathsf{K}_i[\lambda^j_\gamma]\Big)\oplus\lambda^i_\gamma\Delta_i.$$

Thus, we conclude that $\mathsf{L}^i_{\gamma,0}\oplus\mathsf{L}^i_{\gamma,\Lambda_\gamma}$ is equal to:

$$\begin{aligned}& = \Lambda_\alpha\Lambda_\beta\Delta_i\oplus\Lambda_\alpha\lambda_\beta\Delta_i\oplus\Lambda_\beta\lambda_\alpha\Delta_i\oplus\Big(\textstyle\bigoplus_{j\neq i}\mathsf{K}_i[\lambda^j_{\alpha\beta}]\oplus\textstyle\bigoplus_{j\neq i}\mathsf{M}_i[\lambda^j_{\alpha\beta}]\oplus\lambda^i_{\alpha\beta}\Delta_i\Big) \\ & \quad \oplus\Big(\textstyle\bigoplus_{j\neq i}\mathsf{K}_i[\lambda^j_\gamma]\oplus\textstyle\bigoplus_{j\neq i}\mathsf{M}_i[\lambda^j_\gamma]\oplus\lambda^i_\gamma\Delta_i\Big) \\ & = \Lambda_\alpha\Lambda_\beta\Delta_i\oplus\Lambda_\alpha\lambda_\beta\Delta_i\oplus\Lambda_\beta\lambda_\alpha\Delta_i\oplus\lambda_{\alpha\beta}\Delta_i\oplus\lambda_\gamma\Delta_i \\ & = \Lambda_\alpha\Lambda_\beta\Delta_i\oplus\Lambda_\alpha\lambda_\beta\Delta_i\oplus\Lambda_\beta\lambda_\alpha\Delta_i\oplus\lambda_\alpha\lambda_\beta\Delta_i\oplus\lambda_\gamma\Delta_i \\ & = \big((\Lambda_\alpha\oplus\lambda_\alpha)\wedge(\Lambda_\beta\oplus\lambda_\beta)\oplus\lambda_\gamma\big)\Delta_i = \Lambda_\gamma\Delta_i,\end{aligned}$$

where it is easy to verify that $\lambda_{\alpha\beta} = \lambda_\alpha\cdot\lambda_\beta$ according to the Beaver triples. This means that according to P_i's definition of $\mathsf{L}_{\gamma,\Lambda_\gamma}$, the label evaluated by P_1 is always correct. The public value is correct, since $\mathsf{lsb}(\Delta_2) = 1$ and $b_\gamma\oplus\mathsf{lsb}(\mathsf{L}^2_{\gamma,\Lambda_\gamma})$ is equal to:

$$\mathsf{lsb}(\mathsf{L}^2_{\gamma,0})\oplus\mathsf{lsb}(\mathsf{L}^2_{\gamma,\Lambda_\gamma}) = \mathsf{lsb}(\mathsf{L}^2_{\gamma,0}\oplus\mathsf{L}^2_{\gamma,\Lambda_\gamma}) = \mathsf{lsb}(\Lambda_\gamma\Delta_2) = \Lambda_\gamma.$$

□

Lemma 9. *Let $\mathcal{A}$ be a PPT adversary who corrupts a subset of parties such that $P_1 \in A$ is corrupted. Either the execution of protocol Π_{mpc} aborts, or $\mathcal{A}$ learns at most one of two garbled labels for any wire and honest party, except with probability at most $q/2^{\kappa-1}$, where q is the number of H queries.*

Proof. Clearly, adversary $\mathcal{A}$ learns both garbled labels from some honest party $P_i \notin A$ for some wire if and only if $\mathcal{A}$ learns the global key Δ_i. Thus, we only need to prove the probability that the protocol execution does not abort and $\mathcal{A}$ learns Δ_i is at most $q/2^{\kappa-1}$. If $\mathcal{A}$ succeeds to guess c_i bits of Δ_i for some $c_i \in [\kappa]\cup\{0\}$ via the (leak) command of $\mathcal{F}_{\mathsf{prep}}$, then the real protocol execution will abort except with probability $1/2^{c_i}$.

Note that all the MACs received by $\mathcal{A}$ from $\mathcal{F}_{\mathsf{prep}}$ do not include any information on Δ_i, as the local keys are uniformly random. Therefore, only the garbled tables generated by P_i may include the information of Δ_i. In the half-gates garbled rows, Δ_i is encrypted by both garbled labels, and thus is known by $\mathcal{A}$ if and only if $\mathcal{A}$ has queried both garbled labels to random oracle H. Besides, in the garbled rows $\{G^{i,j}_{w,00}, G^{i,j}_{w,01}, G^{i,j}_{w,10}, G^{i,j}_{w,11}\}_{j \neq i,1}$ for $w \in \mathcal{W}$ computed by P_i, the information of Δ_i is only available in the computations $\mathsf{H}(\mathsf{L}^i_{\alpha,u}, \mathsf{L}^i_{\beta,v}, \gamma, j)$ for $(u,v) \in \{0,1\}^2 \backslash \{(\Lambda_\alpha, \Lambda_\beta)\}$, $\gamma \in \mathcal{W}$ and $j \neq i, 1$, and thus $\mathcal{A}$ can only obtain Δ_i if and only if it makes the queries including both garbled labels for some wire to random oracle H.

In both cases, the only way that $\mathcal{A}$ learns Δ_i is to make queries to random oracle H. As a result, the probability, that both garbled labels for some wire have been queried to H by $\mathcal{A}$ (i.e., Δ_i is learned by $\mathcal{A}$), is bounded by $q/2^{\kappa-1-c_i}$. Overall, with probability at most $1/2^{c_i} \cdot q/2^{\kappa-1-c_i} = q/2^{\kappa-1}$, the protocol does not abort and $\mathcal{A}$ learns Δ_i (thus both garbled labels for some wire). □

Lemma 10. *For each $i \in [n]$, let $x^i_w \overset{\mathsf{def}}{=} \Lambda_w \oplus \lambda_w$ for each $w \in \mathcal{I}_i$, where Λ_w is what P_i sends in Step 9a of protocol Π_{mpc} and λ_w is from $\mathcal{F}_{\mathsf{prep}}$. If any PPT adversary $\mathcal{A}$ corrupts a set of parties such that $P_1 \notin A$ is honest, then either P_1 aborts, or P_1 outputs $y^1 = f_1(x^1, \ldots, x^n)$ with probability at least $1 - (|\mathcal{C}| + q + 2)/2^\kappa$, where $\mathbf{H}$ is $|\mathcal{C}|/2^\kappa$-almost universal, $\mathcal{A}$ makes at most q queries to H and f_1 denotes the P_1's output on multi-output function f.*

Proof. After Step 10, P_1 obtains a set of public values for all wires in the circuit $\mathcal{C}$. In the following, we will prove that if these public values are not correct, then P_1 will abort with probability $1 - (|\mathcal{C}| + 1)/2^\kappa$, where recall that we use a polynomial hash to instantiate almost universal hash function $\mathbf{H}$.

We first prove that for each $w \in \mathcal{W}$, we have $t_w = 0$. For each AND gate $(\alpha, \beta, \gamma, \wedge)$, from the definition of t^i_γ for $i \in [n]$, we have

$$\begin{aligned}
\textstyle\bigoplus_{i\in[n]} t^i_\gamma &= \Lambda_\alpha \cdot \Lambda_\beta \oplus \Lambda_\gamma \oplus \Lambda_\alpha \cdot \left(\textstyle\bigoplus_{i\in[n]} \lambda^i_\beta\right) \oplus \Lambda_\beta \cdot \left(\textstyle\bigoplus_{i\in[n]} \lambda^i_\alpha\right) \oplus \left(\textstyle\bigoplus_{i\in[n]} \lambda^i_{\alpha\beta}\right) \oplus \left(\textstyle\bigoplus_{i\in[n]} \lambda^i_\gamma\right) \\
&= \Lambda_\alpha \cdot \Lambda_\beta \oplus \Lambda_\gamma \oplus \Lambda_\alpha \cdot \lambda_\beta \oplus \Lambda_\beta \cdot \lambda_\alpha \oplus \lambda_\alpha \cdot \lambda_\beta \oplus \lambda_\gamma \\
&= (\Lambda_\alpha \oplus \lambda_\alpha) \wedge (\Lambda_\beta \oplus \lambda_\beta) \oplus (\Lambda_\gamma \oplus \lambda_\gamma) = t_\gamma.
\end{aligned}$$

According to the definition of $\{\mathsf{M}_1[t^i_w]\}_{i\neq 1}$ and $\mathsf{M}_1[t^1_w]$ for $w \in \mathcal{W}$, we have the following:

$$\begin{aligned}
\sum_{i=1}^n \mathsf{M}_1[t^i_w] &= \sum_{i\neq 1} \mathsf{K}_1[t^i_w] + t^1_w \Delta_1 + \sum_{i \neq 1} \mathsf{M}_1[t^i_w] \\
&= \sum_{i\neq 1} (\mathsf{K}_1[t^i_w] + \mathsf{M}_1[t^i_w]) + t^1_w \Delta_1 \\
&= \sum_{i=1}^n t^i_w \Delta_1 = t_w \Delta_1.
\end{aligned}$$

For each $i \neq 1$, we use z_i to denote the correct value computed with $\mathbf{H}$ and the MACs held by P_i, and $\hat{z}_i$ to denote the value sent by a malicious party P_i. Thus, $\sum_{i\neq 1} \hat{z}_i = \sum_{i \neq 1} z_i + e$, where e is an adversarily chosen error. Note that z_1 is correct, as P_1 is honest. Since $\mathbf{H}$ is additively homomorphic, we have that

$$\begin{aligned}
\sum_{i=1}^n z_i &= \sum_{i=1}^n \mathbf{H}\left(\left\{\mathsf{M}_1[t^i_w]\right\}_{w\in\mathcal{W}}\right) \\
&= \mathbf{H}\left(\left\{\sum_{i=1}^n \mathsf{M}_1[t^i_w]\right\}_{w\in\mathcal{W}}\right) \\
&= \mathbf{H}\left(\left\{t_w \Delta_1\right\}_{w\in\mathcal{W}}\right) = \mathbf{H}(\{t_w\}_{w\in\mathcal{W}}) \cdot \Delta_1.
\end{aligned}$$

Thus, $\mathbf{H}(\{t_w\}_{w\in\mathcal{W}}) \cdot \Delta_1 = e$, as $\sum_{i\neq 1} \hat{z}_i = 0$ if P_1 does not abort.

Below, we analyze the probability that $\mathbf{H}(\{t_w\}_{w\in\mathcal{W}}) \neq 0$ but P_1 does not abort. We assume that the adversary $\mathcal{A}$ leaks c_1 bits of Δ_1 for some $c_1 \in [\kappa] \cup \{0\}$ by the (leak) command of $\mathcal{F}_{\mathsf{prep}}$. In this case, the real protocol execution will abort except with probability 2^{-c_1}. Then the remaining $\kappa - c_1$ bits of Δ_1 are uniformly random from the adversary's view. Thus, e and $\{t_w\}_{w\in\mathcal{W}}$ are independent of the unknown $\kappa - c_1$ bits of Δ_1. As P_1 is honest, linear hash function $\mathbf{H}$ defined by a random seed χ is independent of Δ_1. Therefore, under the condition that c_1 bits of Δ_1 have already been leaked, the probability that $\Delta_1 = \mathbf{H}(\{t_w\}_{w\in\mathcal{W}})^{-1} \cdot e$ is at most $2^{c_1-\kappa}$. Overall, with probability $2^{-c_1} \cdot 2^{c_1-\kappa} = 2^{-\kappa}$, P_1 does not abort and $\mathbf{H}(\{t_w\}_{w\in\mathcal{W}}) \neq 0$. Since $\{t_w\}_{w\in\mathcal{W}}$ are independent of $\mathbf{H}$ and $\mathbf{H}$ is $|\mathcal{C}|/2^{\kappa}$-almost universal, the probability that there exists one $w \in \mathcal{W}$ such that $t_w \neq 0$ is at most $|\mathcal{C}|/2^{\kappa}$. Overall, with probability at least $1 - (|\mathcal{C}|+1)/2^{\kappa}$, we have $t_w = 0$ for all $w \in \mathcal{W}$.

Below, we prove by induction that for each wire w, public value Λ_w is correct.

Base step: The public values for all circuit-input wires are correct, according to how x_w^i is defined for each $i \in [n], w \in \mathcal{I}_i$.

Induction step: It is easy to verify that the public values for the output wires of XOR gates are correct. So, we will focus on each AND gate $(\alpha, \beta, \gamma, \wedge)$. According to the induction hypothesis, we have that P_1 holds correct public values Λ_α and Λ_β. Recall that the correctness of public value Λ_γ is checked by computing the following value:

$$t_\gamma = (\Lambda_\alpha \oplus \lambda_\alpha) \wedge (\Lambda_\beta \oplus \lambda_\beta) \oplus (\Lambda_\gamma \oplus \lambda_\gamma).$$

From $t_\gamma = 0$, we have $\Lambda_\gamma = (\Lambda_\alpha \oplus \lambda_\alpha) \wedge (\Lambda_\beta \oplus \lambda_\beta) \oplus \lambda_\gamma$. According to the correctness of Λ_α and Λ_β, $\Lambda_\alpha \oplus \lambda_\alpha$ and $\Lambda_\beta \oplus \lambda_\beta$ are the correct actual values for input wires α and β respectively. Therefore, Λ_γ is correct.

If P_1 does not abort in Step 13 of protocol Π_{mpc}, the probability that there exists a corrupt party P_j flipping its share λ_w^j for some $w \in \mathcal{O}_1$ is $(q+1)/2^{\kappa}$, according to Lemma 1. From the above proof by induction, we have that public value Λ_w is correct for each $w \in \mathcal{O}_1$, except with probability $(|\mathcal{C}|+1)/2^{\kappa}$. In conclusion, if P_1 does not abort, $y_w^1 = \Lambda_w \oplus \lambda_w$ is correct for each $w \in \mathcal{O}_1$, except with probability $(|\mathcal{C}|+q+2)/2^{\kappa}$. □

Lemma 11. *For every PPT adversary $\mathcal{A}$ corrupting a subset of parties, every honest party $P_i \notin A$ either aborts, or outputs $y^i = f_i(x^1, \ldots, x^n)$ with probability at least $1 - 3q/2^{\kappa}$, where f_i denotes the output of function f to P_i and q is the number of* H *queries.*

Proof. We first prove that $P_i \notin A$ either aborts or obtains the correct public values in Step 11 of protocol Π_{mpc}, even if P_1 is corrupted by $\mathcal{A}$. Let $\{\Lambda'_w\}_{w\in\mathcal{W}}$ be the public values received by P_i in Step 11 when P_1 is corrupted, and $\{\Lambda_w\}_{w\in\mathcal{W}}$ be the correct public values that should be sent by an honest P_1. Below, we analyze the probability that there exists some $w \in \mathcal{W}$ such that $\Lambda'_w \neq \Lambda_w$. In Step 11, $\mathcal{A}$ on behalf of P_1 sends a value h'_i to P_i. If $P_i \notin A$ does not abort, then we have that

$$h'_i = \mathsf{H}\left(\left\{\mathsf{L}_{w,0}^i \oplus \Lambda'_w \Delta_i\right\}_{w\in\mathcal{W}}\right).$$

Since H is a random oracle, the probability that $\mathcal{A}$ finds a target collision is $q/2^{\kappa}$. Therefore, with probability $1 - q/2^{\kappa}$, $\mathsf{L}_{w,\Lambda'_w}^i = \mathsf{L}_{w,0}^i \oplus \Lambda'_w \Delta_i$ for each $w \in \mathcal{W}$ is learned by $\mathcal{A}$. In addition, $\mathcal{A}$ has learned $\mathsf{L}_{w,\Lambda_w}^i = \mathsf{L}_{w,0}^i \oplus \Delta_w \Delta_i$ by evaluating the circuit on behalf of P_1. If $\Lambda'_w \neq \Lambda_w$ for some $w \in \mathcal{O}_i$, then $\mathcal{A}$ learns both garbled labels $\mathsf{L}_{w,0}^i$ and $\mathsf{L}_{w,1}^i$ for the wire w. By Lemma 9, this happens with probability at most $q/2^{\kappa-1}$. Overall, except with probability at most $3q/2^{\kappa}$, the public values on all wires in $\mathcal{W}$ received by P_i are correct, if P_i does not abort. Together with that the public values for all circuit-input wires are correct and the public values on the output wires of XOR gates are correct by induction, we obtain that the public values on all wires in the circuit are correct, except with probability at most $3q/2^{\kappa}$.

Based on the proof of Lemma 1, the probability that P_i does not abort in Step 13 and there exists a malicious party P_j flipping its share λ_w^j for some $w \in \mathcal{O}_i$ is bounded by $3q/2^{\kappa}$. In this probability, the probability that $\mathcal{A}$ finds a target collision for H is bounded by $q/2^{\kappa}$; the probability that the real protocol execution does not abort and adversary $\mathcal{A}$ learns Δ_i is at most $q/2^{\kappa-1}$ from the proof of Lemma 9. Therefore, P_i will obtain a correct wire mask λ_w for each $w \in \mathcal{O}_i$, except with probability at most $3q/2^{\kappa}$.

In conclusion, if P_i does not abort, then $y_w^i = \Lambda_w \oplus \lambda_w$ is correct for each $w \in \mathcal{O}_i$, except with probability at most $3q/2^{\kappa}$. □

E.2 Proof of Theorem 3

Given the Lemmas 8−11, the proof of Theorem 3 is relatively easy. Below, we present the details of the proof.

Theorem 7 (Theorem 3, restated)**.** *Let $f : \{0,1\}^{|\mathcal{I}|} \to \{0,1\}^{|\mathcal{O}|}$ be an n-party functionality. Then protocol Π_{mpc} shown in Figures 6 and 7 securely computes f in the presence of a static malicious adversary corrupting up to $n-1$ parties in the $\mathcal{F}_{\mathsf{prep}}$-hybrid model, where H is a random oracle.*

Proof. Let $\mathcal{A}$ be a PPT adversary who corrupts a subset of parties A. We construct a PPT simulator $\mathcal{S}$, which runs $\mathcal{A}$ as a subroutine, simulates the adversary's view, and has access to an ideal functionality $\mathcal{F}_{\mathsf{mpc}}$ that implements f. Whenever any honest party simulated by $\mathcal{S}$ aborts or $\mathcal{A}$ aborts, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and aborts. The simulator $\mathcal{S}$ is defined as below.

Description of the simulation.

- Initialization: After $\mathcal{A}$ corrupted a subset of parties A, $\mathcal{S}$ corrupts the same parties in the ideal world, and internally emulates an execution of the honest parties running Π_{mpc} with $\mathcal{A}$.

- Preprocessing: $\mathcal{S}$ emulates the functionality $\mathcal{F}_{\mathsf{prep}}$, and records all the values from adversary $\mathcal{A}$. Simulator $\mathcal{S}$ acts as every honest party $P_i \notin A$ and simulates honestly the execution of P_i in function-(in)dependent phases.

- Online: $\mathcal{S}$ simulates honestly the execution of honest parties, with the following exceptions:
  - For every honest party $P_i \notin A$, $\mathcal{S}$ adopts $x^i := 0^{|I_i|}$ as P_i's input, and broadcasts $\Lambda_w := \lambda_w$ to all parties for each circuit-input wire $w \in \mathcal{I}_i$.
  - For each corrupt party $P_i \in A$, for every $w \in \mathcal{I}_i$, $\mathcal{S}$ receives a public value Λ_w from $\mathcal{A}$, and computes $x_w^i := \Lambda_w \oplus \lambda_w$ as an input bit of P_i.
  - For every corrupt party $P_i \in A$, $\mathcal{S}$ sends (input, x^i) on behalf of P_i to $\mathcal{F}_{\mathsf{mpc}}$, and receives an output y^i. $\mathcal{S}$ computes
  $$(\tilde{y}^1, \ldots, \tilde{y}^n) := f(\tilde{x}^1, \ldots, \tilde{x}^n),$$
  where $\{\tilde{x}^i := 0^{|I_i|}\}_{i \notin A}$ and $\{\tilde{x}^i := x^i\}_{i \in A}$. Then $\mathcal{S}$ chooses any $j^* \notin A$, and then for each $i \in A, w \in \mathcal{O}_i$, defines $\tilde{\lambda}_w^{j^*} := \lambda_w^{j^*} \oplus y_w^i \oplus \tilde{y}_w^i$ and computes $\mathsf{M}_i[\tilde{\lambda}_w^{j^*}] := \mathsf{M}_i[\lambda_w^{j^*}] \oplus (\tilde{\lambda}_w^{j^*} \oplus \lambda_w^{j^*})\Delta_i$. For each $i \in A$, $\mathcal{S}$ acts as honest party P_{j^*} and opens $\{\tilde{\lambda}_w^{j^*}\}_{w \in \mathcal{O}_i}$ to corrupt party P_i in the amortized way.

Based on Lemmas 8–11 in the previous section, we prove that the real protocol execution is indistinguishable from the ideal world execution by a sequence of games.

$\mathbf{Hybrid_0}$. This is the same as the real protocol execution shown in Figures 6 and 7, where the actual inputs $\{x^i\}_{i \notin A}$ are used for honest parties.

Hybrid$_1$. This is the same as **Hybrid$_0$**, except that $\mathcal{S}$ plays the role of honest parties $\{P_i\}_{i \notin A}$.

Hybrid$_1$ is essentially the same as **Hybrid$_0$**.

Hybrid$_2$. This is the same as **Hybrid$_1$**, except that a) for each $i \in A, w \in \mathcal{I}_i$, $\mathcal{S}$ receives a public value Λ_w from $\mathcal{A}$ and computes $x_w^i := \Lambda_w \oplus \lambda_w$; b) for each $i \in A$, $\mathcal{S}$ sends (input, x^i) on behalf of P_i to $\mathcal{F}_{\mathsf{mpc}}$ and receives an output y^i.

The distributions on the view of adversary $\mathcal{A}$ in **Hybrid$_1$** and **Hybrid$_2$** are identical. If P_1 is honest, then the outputs obtained by P_1 in two hybrids are the same except with probability at most $(|\mathcal{C}|+q+2)/2^\kappa = \mathsf{negl}(\kappa)$ from Lemma 8 and Lemma 10, where q is an upper bound of the number of H queries. If P_i is honest for each $i \notin A, i \neq 1$, then the outputs obtained by P_i in two hybrid games are the same except with probability at most $3q/2^\kappa = \mathsf{negl}(\kappa)$ by Lemma 11. Therefore, the distributions in **Hybrid$_1$** and **Hybrid$_2$** are indistinguishable, except with probability $\mathsf{negl}(\kappa)$.

Hybrid$_3$. This is the same as **Hybrid$_2$**, except that simulator $\mathcal{S}$ executes as follows:

1. Use $\{x^i = 0^{|I_i|}\}_{i \notin A}$ as the inputs of honest parties in Step 9 of protocol Π_{mpc}.
2. Compute $(\tilde{y}^1, \ldots, \tilde{y}^n) := f(\tilde{x}^1, \ldots, \tilde{x}^n)$, where $\tilde{x}^i := 0^{|I_i|}$ for each $i \notin A$ and $\tilde{x}^i := x^i$ for each $i \in A$.
3. Choose any $j^* \notin A$, and for each $i \in A, w \in \mathcal{O}_i$ define $\tilde{\lambda}_w^{j^*} := \lambda_w^{j^*} \oplus y_w^i \oplus \tilde{y}_w^i$. Then compute $\mathsf{M}_i[\tilde{\lambda}_w^{j^*}] := \mathsf{M}_i[\lambda_w^{j^*}] \oplus (\tilde{\lambda}_w^{j^*} \oplus \lambda_w^{j^*})\Delta_i$.
4. For each $i \in A$, act as honest party P_{j^*} and open $\{\tilde{\lambda}_w^{j^*}\}_{w \in \mathcal{O}_i}$ to corrupt party P_i in the amortized way.

We first prove that for every honest party P_i, its share λ_w^i for each wire w in the circuit is uniformly random and kept secret in $\mathcal{A}$'s view, before these shares are revealed in the phases of input and output processing. Here we do not consider the circuit-input wires associated with other parties' inputs, as the corresponding shares are set as 0. If $i = 1$, it is easy to see that the P_1's shares for all wires are kept secret in the information-theoretic sense. If $i \neq 1$, we show that P_i's shares are computationally hidden, even if P_1 is corrupted. For each XOR gate $(\alpha, \beta, \gamma, \oplus)$, $\lambda_\gamma^i = \lambda_\alpha^i \oplus \lambda_\beta^i$ is kept unknown for $\mathcal{A}$, if at least one of λ_α^i and λ_β^i is kept secret. Thus, we focus on each AND gate $(\alpha, \beta, \gamma, \wedge)$. The half-gates garbled rows $\mathcal{G}_{\gamma,0}^i$ and $\mathcal{G}_{\gamma,1}^i$ for $\gamma \in \mathcal{W}$ are encrypted by both garbled labels for input wires. Therefore, $\mathcal{A}$ are still unknown for P_i's shares λ_α^i and λ_β^i on input wires α and β, unless it learns Δ_i. From Lemma 9, this occurs with probability at most $q/2^{\kappa-1} = \mathsf{negl}(\kappa)$. Besides, each garbled row $G_{\gamma,uv}^{i,j}$ for each $\gamma \in \mathcal{W}$, $u, v \in \{0, 1\}$ and $j \neq i, 1$ is encrypted using different combinations of $\mathsf{L}_{\alpha,0}^i, \mathsf{L}_{\alpha,1}^i$ and $\mathsf{L}_{\beta,0}^i, \mathsf{L}_{\beta,1}^i$. To open at least two garbled rows, $\mathcal{A}$ needs to learn both garbled labels for some wire. From Lemma 9, this happens with probability at most $q/2^{\kappa-1} = \mathsf{negl}(\kappa)$. Therefore, $\mathcal{A}$ does not learn the shares λ_α^i and λ_β^i for the input wires. In the process of checking public values, if P_i with $i \neq 1$ does not abort, the public values on the output wires of all AND gates are correct except with probability at most $3q/2^\kappa = \mathsf{negl}(\kappa)$, according to the proof of Lemma 11. Therefore, the value z_i sent by P_i does not reveal its shares for each AND gate $(\alpha, \beta, \gamma, \wedge)$, as $\lambda_{\alpha\beta}^i$ is uniformly random and masks P_i's shares.

For each $i \notin A, w \in \mathcal{I}_i$, we have proved that λ_w^i is uniformly random and unknown for $\mathcal{A}$. Therefore, the distributions of the public values $\{\Lambda_w\}_{w \in \bigcup_{i \notin A} \mathcal{I}_i}$ in **Hybrid$_2$** and **Hybrid$_3$** are both independently random, and thus are exactly the same. For each wire w associated with the outputs of corrupt parties, $\mathcal{A}$ does not know the share $\lambda_w^{j^*}$ of $P_{j^*} \notin A$, and both $\lambda_w^{j^*}$ and $\tilde{\lambda}_w^{j^*}$ are uniformly random. Therefore, for each $i \in A, w \in \mathcal{O}_i$, $\tilde{\lambda}_w^{j^*}$ sent by $\mathcal{S}$ in **Hybrid$_3$** has the same distribution as $\lambda_w^{j^*}$ sent by honest party P_{j^*} in **Hybrid$_2$**.

For each $w \in \mathcal{W}$, the public value Λ_w is uniformly random and has the same distribution in **Hybrid$_2$** and **Hybrid$_3$**, as $\{\lambda_w^i\}_{i \notin A}$ are uniformly random and not known to $\mathcal{A}$. If $P_1 \in A$, then P_1 is able to learn only one garbled label for each wire except with probability at most $q/2^{\kappa-1} = \mathsf{negl}(\kappa)$ by Lemma 9. Thus, P_1 can open only one of four garbled rows $G^{i,j}_{\gamma,00}, G^{i,j}_{\gamma,01}, G^{i,j}_{\gamma,10}, G^{i,j}_{\gamma,11}$ for each $\gamma \in \mathcal{W}$, $i \notin A$ and $j \neq i, 1$. In two hybrids, the distribution of garbled rows evaluated by corrupt party P_1 is indistinguishable, as the distribution of public values $\{\Lambda_w\}_{w \in \mathcal{W}}$ is the same. Moreover, the garbled labels obtained by P_1 are indistinguishable in two hybrids.

Based on the proof of Lemma 10, if honest party $P_1 \notin A$ does not abort in Step 12 of protocol Π_{mpc}, $\Lambda_w = \tilde{y}^i_w \oplus (\bigoplus_{j \in [n]} \lambda^j_w)$ for each $w \in \mathcal{O}_i$ except with probability at most $(|\mathcal{C}|+1)/2^\kappa = \mathsf{negl}(\kappa)$. Therefore, for each $i \in A$ and $w \in \mathcal{O}_i$, $\Lambda_w \oplus (\bigoplus_{j \neq j^*} \lambda^j_w) \oplus \tilde{\lambda}^{j^*}_w = \Lambda_w \oplus (\bigoplus_{j \in [n]} \lambda^j_w) \oplus y^i_w \oplus \tilde{y}^i_w = y^i_w$, which means that $\mathcal{A}$ will obtain the correct output. If $P_1 \in A$, $\mathcal{A}$ will also get the correct output for each $i \in A$, due to the setting of the shares $\{\tilde{\lambda}^{j^*}_w\}_{w \in \mathcal{O}_i}$ of honest party P_{j^*}.

In conclusion, $\mathbf{Hybrid_3}$ is indistinguishable from $\mathbf{Hybrid_2}$, except with probability $\mathsf{negl}(\kappa)$. □