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

September 27, 2019

Abstract

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

We improve the protocol for generating authenticated AND triples, which plays a crucial role in many recent works.

– We propose a multi-party authenticated bit protocol from bare IKNP OT extension, allowing us to 1) reduce the communication by $\approx$ 24% with a small harmless leakage, and 2) eliminate many computation-heavy procedures like bit-matrix multiplications and consistency checks. This improvement also applies to the two-party setting.

– We reduce the cost of consistency check for multi-party authenticated shares by ρ times where ρ is the statistical security parameter, and also cut the number of hash function calls per party by a factor of $2\times$ when computing leaky authenticated AND triples.

We further 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 AND 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 size of garbled tables by about 4ρ bits per AND gate, by using an almost universal hash function to perform the circuit authentication in a batch. Prior solution with similar efficiency is only applicable in the two-party setting.

Along with other optimization, we reduce the communication complexity of the whole protocol (including both function-independent and function-dependent preprocessing phases that require the most resources) by $\approx$ 24% and significantly improve the overall computational efficiency.

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 corruptions in the presence of malicious adversaries.

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 (Ind.):** The parties need not know the function to be evaluated or their inputs, besides an upper bound on the number of gates.

   In many state-of-the-art protocols, the primary job of this phase is to generate random multi-party authenticated AND triples (to be discussed later). It requires the most computation and communication resources.

2. **Function-dependent phase (Dep.):** The parties now know the function being computed, but need not know their inputs.

   This phase usually involves generating a multi-party garbled circuit, which may be either asymmetric [WRK17b] or symmetric [HSS17] depending on the number of parties who can evaluate the garbled circuit.

3. **Online phase:** The parties evaluate the function on their inputs.

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. See below.

1. The communication overhead to obtain malicious security is still high. For example, the best-known constant-round maliciously secure two-party computation (2PC) protocol [KRRW18] still requires sending ≈ 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.

2. The computational overhead of existing maliciously secure protocols is surprisingly higher than commonly thought. For example, the most efficient implementation [WRK17a] for 2PC can generate about $833K$ authenticated AND triples per second under the 10 Gbps bandwidth network and a 36-core CPU. However, if the network was fully used, we would expect at least $4,250K$ 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 checks for consistency need to be performed between all pairs of the parties. For example, with eight parties, the implementation from Wang et al. [WRK17b], which was benchmarked using the same hardware as above, can compute about $68K$ multi-party authenticated AND triples per second. The speed would be $510K$ authenticated AND triples per second, if the 10 Gbps bandwidth network was fully utilized, a performance gap of $7.5\times$.

   One of the goals of this work is to figure out the source of these inefficiencies and to fix them with enhanced protocols. See Section 2 for our observation and analysis.

[1]Since every authenticated AND triple in their protocol takes ≈ 294 bytes of communication, 10 Gbps bandwidth can support around 4.25×10^6 triples.

These computation and communication overhead 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 authenticated garbling paradigm for constant-round multi-party computation in the malicious setting, tolerating an arbitrary number of corruptions. To fully explore potential improvements, we start from the most basic building blocks, even within the protocols for maliciously secure OT extension. We summarize our contribution below and provide an intuition of our ideas in Section 2.

First, we design a new authenticated AND triple (aAND) protocol from scratch with improved efficiency. Our improvements can be directly applied right out of the box to many MPC protocols that need authenticated bits/shares/AND triples [WRK17a, WRK17b, HSS17, KRRW18, ZCSH18, RW19, AOR+19, 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 consistency checks for malicious security and input consistency. As an evidence, two-party authenticated bit is about $3\times$ slower than ordinary maliciously secure OT extension even when running locally [Rin].

  We propose a new multi-party authenticated bit protocol directly based on the IKNP OT extension [IKNP03] 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 2.1 for more elaboration.

– **Improved multi-party authenticated share.** Multi-party authenticated share (aShare) is another key building block towards aAND (See Section 2.2). The previous protocol [WRK17b] for multi-party authenticated shares needs to repeat a checking procedure by ρ times, to boost the soundness error from $1/2$ to $1/2^\rho$. 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 $2\times$ when computing leaky AND triples and show that the security preserves even given the leakage introduced in aBit as above. See Section 2.3 for more details.

By applying the above optimizations, we can reduce the communication cost of authenticated AND triple generation by $\approx 24\%$ when $\rho = 40$ ($\approx 28\%$ if $\rho = 64$), where this cost dominates the communication cost of the whole MPC protocol. The computational cost is also improved significantly.

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

– **Towards multi-party half-gates.** Although it is known how to distributively compute half-gates scheme in the two-party setting [KRRW18], multi-party setting is completely open. We *partially* apply the idea of half-gates [ZRE15] in the multi-party setting and reduce the size of garbled tables from each garbler by 2κ bits per AND gate. Our technique here is also applicable in the semi-honest setting [BLO16]. See Section 2.4 for a high-level description.

– **Improved circuit authentication.** Katz et al. [KRRW18] proposed an efficient way to authenticate distributed garbled circuits by using amortized MAC checks. 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 batch circuit authentication procedure based on almost universal linear hash functions [CDD+16]. The resulting solution improves the communication from 4ρ bits to 1 bit per AND gate. See Section 2.5 for more discussion.

– **Other optimization.** We also reduce the cost of input processing by a factor of n, where n is the number of parties.

As a result, for example in the three-party setting, our optimizations result in more than $1.5\times$ improvement in communication for function-dependent phase.

1.2 Organization

In Section 2, we introduce important concepts and building blocks needed for our main 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 basic concepts and some 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. We describe in Section A more commonly known preliminaries, including the communication model that our protocols work, 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.

Notation. Throughout this paper, we use κ and ρ to denote the computational and statistical security parameters respectively. We use $:=$ to denote assignment, and $a \leftarrow A$ to denote sampling a uniformly at random from a set A. 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. Sometimes, we view bit-strings in $\{0,1\}^k$ as vectors in $\mathbb{F}_2^k$ and vice versa, and denote exclusive-or by "$\oplus$" or "$+$". We denote by $\mathbf{x} \in \mathbb{F}_2^k$ a vector, and by $\mathbf{0} \in \mathbb{F}_2^k$ a zero vector. By $\mathbf{x}[k]$, we denote the k-th component of vector $\mathbf{x}$. We also view vectors in $\mathbb{F}_2^k$ as elements in $\mathbb{F}_{2^k}$, and vice versa. We will use a hash function $\mathsf{H} : \{0,1\}^* \rightarrow \{0,1\}^\kappa$ modeled as a random oracle.

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 wires 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 for a circuit $\mathcal{C}$. We denote by $P_1, \ldots, P_n$ the parties. 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., $|\mathcal{I}_1| = \cdots = |\mathcal{I}_n| = |\mathcal{I}|$ and $|\mathcal{O}_1| = \cdots = |\mathcal{O}_n| = |\mathcal{O}|$.

In this paper, we consider a static, malicious adversary who can corrupt up to $n-1$ out of n parties. We use $\mathcal{M} \subset [n]$ to denote the set of all corrupt parties, and $\mathcal{H} = [n] \backslash \mathcal{M}$ to denote the set of all honest parties. All our protocols allow abort, and are provably secure in the stand-alone simulation-based security model [Gol04].

2.1 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 $\{\mathsf{M}_j[b] := \mathbf{0}\}_{j\neq i}$; 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.

The above representation assumes that P_i uses a single global key Δ_i. When other global key Φ_i is used, we will explicitly add a subscript to the representation. That is, we will use $\mathsf{K}_i[x]_{\Phi_i}$ and $\mathsf{M}_i[x]_{\Phi_i} = \mathsf{K}_i[x]_{\Phi_i} \oplus x\Phi_i$ to denote the local key and MAC respectively in this case.

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] followed by a KOS correlation check [KOS15] for consistency. 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 between two parties, 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 in the first step 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 $\max_c\{2^{-c} \times 2^{-(\kappa-c)}\} = 2^{-\kappa}$, which is the same with 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 wrong. The protocol consists of only two steps: 1) P_i executes the IKNP OT extension protocol [IKNP03] 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.

2.2 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 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 $\langle x \rangle = \big([x^1]_1, \dots [x^n]_n\big)$ 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 $\langle x \rangle \oplus b = \big([x^1]_1 \oplus b, [x^2]_2, \dots, [x^n]_n\big)$, and define $b\langle x \rangle$ to be equal to $\langle 0 \rangle = ([0]_1, \dots, [0]_n)$ 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 process, 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 field 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 guarantees the consistency of global keys.

The checking procedure can also be efficiently implemented given hardware support for finite field multiplication. See Section 3.2 for the detailed protocol.

2.3 Improved Authenticated AND triples

The protocol for leaky authenticated AND triples is to generate a random authenticated AND triple $(\langle x \rangle, \langle y \rangle, \langle z \rangle)$ with one caveat that the adversary can choose to guess the share x^i of an honest party. 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 H 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 Section C.1.

2.4 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:

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

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-gate technique can be applied to the above two-party distributed garbling. Although P_2 cannot compute G_0, G_1 and $\mathsf{L}_{\alpha,0}$ 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 free-XOR 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:

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

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

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

   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:

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

The correctness holds because

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

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.

2.5 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 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 $\mathbf{H}$, which further reduces the communication per garbler to κ bits. Specifically, each garbler P_i $(i \neq 1)$ sends $\mathbf{c}_i = \mathbf{H}(\{\mathsf{M}_1[t^i_w]\}_{w \in \mathcal{W}}) \in \mathbb{F}_2^\kappa$ to P_1, and P_1 computes $\mathsf{M}_1[t^1_w] := \bigoplus_{i \neq 1} \mathsf{K}_1[t^i_w] \oplus t^1_w \Delta_1$ for each wire $w \in \mathcal{W}$. For each $w \in \mathcal{W}$, $t_w = 0$ if and only if

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

Then, P_1 computes $\mathbf{c}_1 := \mathbf{H}(\{\mathsf{M}_1[t^1_w]\}_{w \in \mathcal{W}})$, and checks that $\bigoplus_{i \in [n]} \mathbf{c}_i = \mathbf{0}$. By the XOR-homomorphic property of $\mathbf{H}$, we have

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

From the almost-universal property, we have that $\bigoplus_{i \in [n]} \mathsf{M}_1[t^i_w] = \mathbf{0}$ for $w \in \mathcal{W}$. We can use a polynomial hash based on GMAC to instantiate the linear hash function $\mathbf{H}$. In particular, the parties use a random seed $\chi \in \mathbb{F}_{2^\kappa}$ to define $\mathbf{H}$. For $i \in [n]$, $\mathbf{c}_i = \mathbf{H}(\{\mathsf{M}_1[t^i_w]\}_{w \in \mathcal{W}}) = \mathbf{H}(\mathsf{M}_1[t^i_{w_1}], \ldots, \mathsf{M}_1[t^i_{w_{|\mathcal{C}|}}])$ can be computed as $\sum_{k=1}^{|\mathcal{C}|} \chi^k \cdot \mathsf{M}_1[t^i_{w_k}] \in \mathbb{F}_{2^\kappa}$, where $w_1, \ldots, w_{|\mathcal{C}|} \in \mathcal{W}$ denote the output wires of all AND gates in the circuit $\mathcal{C}$. When $\kappa = 128$, the finite field multiplication over $\mathbb{F}_{2^\kappa}$ can be performed very efficiently using the native instructions [KOS15, NST17].

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 straightfoward way. We give the detailed protocol in Section 4 and the full proof in Section D.

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

Honest Parties:

1. Upon receiving (init) from all parties, for each $i \in [n]$, sample $\Delta_i \leftarrow \{0,1\}^\kappa$ and send Δ_i to P_i.
2. Upon receiving (aBit, i) from all parties, sample a random bit $\lambda \leftarrow \{0,1\}$ and generate a random authenticated bit $[\lambda]_i = (\lambda, \{\mathsf{M}_j[\lambda]\}_{j \neq i}, \{\mathsf{K}_j[\lambda]\}_{j \neq i})$. Then send $(\lambda, \{\mathsf{M}_j[\lambda]\}_{j \neq i})$ to P_i and $\mathsf{K}_j[\lambda]$ to P_j for each $j \neq i$.
3. Upon receiving (aShare) from all parties, sample a random bit $\lambda \leftarrow \{0,1\}$ and generate a random authenticated share $\langle\lambda\rangle = \big([\lambda^1]_1, \ldots, [\lambda^n]_n\big)$ with $\bigoplus_{i \in [n]} \lambda^i = \lambda$. Distribute $\langle\lambda\rangle$ to all parties, i.e., send $(\lambda^i, \{\mathsf{M}_j[\lambda^i], \mathsf{K}_i[\lambda^j]\}_{j \neq i})$ to P_i for each $i \in [n]$.
4. 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 $(\langle a\rangle, \langle b\rangle, \langle c\rangle)$. Distribute the authenticated shares $\langle a\rangle, \langle b\rangle, \langle c\rangle$ to all parties.

Corrupt Parties: Corrupt parties can choose their own randomness used to define the outputs that they receive from the functionality.

Leakage of global keys: The adversary may input $(\mathsf{leak}, i, S, \{\Delta'[k]\}_{k \in S})$. If P_i is honest, the functionality does the following:

– If there exists some $k \in S$ such that $\Delta'[k] \neq \Delta_i[k]$, the functionality outputs `fail` to all parties and aborts.

– Otherwise, it outputs `success` to the adversary and proceeds as if nothing has happened.

Figure 1: The multi-party preprocessing functionality with weak global keys

2.6 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 Bit2Share. Specifically, $\mathsf{Bit2Share}([\lambda]_i)$ takes as input an authenticated bit $[\lambda]_i = \big(\lambda, \{\mathsf{M}_k[\lambda]\}_{k \neq i}, \{\mathsf{K}_k[\lambda]\}_{k \neq i}\big)$ 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 $\big\{\mathsf{M}_k[\lambda^i] := \mathsf{M}_k[\lambda]\big\}_{k \neq i}$, 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 $\{\mathsf{M}_k[\lambda^j] := \mathbf{0}\}_{k \neq j}$, and P_k defines $\mathsf{K}_k[\lambda^j] := \mathbf{0}$ 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 Multi-Party 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 protocols in Section 2, here we will focus on detailed description of the protocols and their security proofs.

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. Due to space limitation, we present our improved protocol for leaky authenticated AND triples in Section C.1, and that the bucketing technique [NNOB12] for eliminating leakage of partial shares can still be applied in our setting in Section C.2.

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

The functionality is run by a sender P_S and a receiver P_R, and executes as follows:

Initialize: Upon receiving (init, Δ) from P_S where $\Delta \in \{0,1\}^\kappa$, and (init) from P_R, store Δ. Ignore any subsequent init command.

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, the functionality does the following:

– Sample $\mathsf{K}[x_i] \leftarrow \mathbb{F}_2^\kappa$ and 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.

– For each $i \in [\ell]$, send $\mathsf{M}[x_i]$ to P_R and $\mathsf{K}[x_i]$ to P_S.

For each $i \in [\ell]$, a corrupt party can choose its own randomness received from the functionality as follows:

– If P_S is corrupted, it gets to choose $\mathsf{K}[x_i]$, and the functionality sets $\mathsf{M}[x_i] := \mathsf{K}[x_i] + \mathbf{x}_i * \Delta$.

– If P_R is corrupted, it gets to choose $\mathsf{M}[x_i]$, and the functionality sets $\mathsf{K}[x_i] := \mathsf{M}[x_i] + \mathbf{x}_i * \Delta$.

Figure 2: Functionality for correlated OT with errors

Our protocols for authenticated bits, shares and AND triples jointly implement the preprocessing functionality $\mathcal{F}_{\mathsf{prep}}$ as shown in Figure 1. In $\mathcal{F}_{\mathsf{prep}}$, we allow the adversary to make multiple leak queries on the same index $i \in \mathcal{H}$. 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 set $S \subset [\kappa]$ representing the coordinates in which the guessed bits locate. Our MPC protocol with improved multi-party authenticated garbling as shown in next section will work in the $\mathcal{F}_{\mathsf{prep}}$-hybrid model.

3.1 Improved Multi-Party Authenticated Bits

We propose a new protocol Π_{aBit} to generate authenticated bits. Our protocol uses a *correlated OT with errors* functionality $\mathcal{F}_{\mathsf{COTe}}$ [KOS15] shown in Figure 2. In $\mathcal{F}_{\mathsf{COTe}}$, if a receiver P_R is honest, it will input a "monochrome" vector $\mathbf{x}_i = x_i \cdot (1, \ldots, 1)$ for $i \in [\ell]$, 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 will 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 $\mathsf{M}[x_i] = \mathsf{K}[x_i] + x_i \cdot \Delta + \mathbf{e}_i * \Delta$, 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 $\mathcal{F}_{\mathsf{COTe}}$, is described in [Nie07, KOS15]. The protocol is the same as the original IKNP OT extension protocol [IKNP03], 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. Note that the bits sampled by P_i in our protocol are authenticated by *weak* global keys, where the adversary may guess a few bits of the honest parties' global keys. We use a functionality $\mathcal{F}_{\mathsf{aBit}}$ shown in Figure 3 to abstract and define authenticated bits with weak global keys. In the init command of $\mathcal{F}_{\mathsf{aBit}}$, an honest party P_j will input a uniformly random global key Δ_j, while a corrupt 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 $j \in \mathcal{H}$. A correct guess keeps undetected, while an incorrect guess will be caught. While the adversary succeeds to leak c_j bits of Δ_j with probability 2^{-c_j}, the other $\kappa - c_j$ bits of Δ_j remain uniformly random in its view.

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

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

The functionality runs with parties $P_1, \ldots, P_n$, and generates random bits known by a party P_i and authenticated to all other parties.

Honest Parties:

– Upon receiving (init, i) from all parties, the functionality receives $\Delta_j \in \{0,1\}^\kappa$ from P_j for each $j \neq i$ and stores Δ_j.

– Upon receiving (aBit, i, ℓ) from all parties, sample $x_1, \ldots, x_\ell \leftarrow \{0,1\}$ and generate random authenticated bits $\{[x_k]_i\}_{k\in[\ell]}$, and then send them to the parties. In detail, for each $j \neq i$, the functionality samples random $\{\mathsf{K}_j[x_k] \leftarrow \{0,1\}^\kappa\}_{k\in[\ell]}$ and compute $\{\mathsf{M}_j[x_k] := \mathsf{K}_j[x_k] \oplus x_k\Delta_j\}_{k\in[\ell]}$. Then, for each $k \in [\ell]$, then functionality sends $(x_k, \{\mathsf{M}_j[x_k]\}_{j\neq i})$ to P_i and $\mathsf{K}_j[x_k]$ to P_j for $j \neq i$.

Corrupt Parties: For each $k \in [\ell]$, corrupt parties can choose their own randomness used to compute the values that they receive from the functionality:

– If P_i is corrupted, it gets to choose $(x_k, \{\mathsf{M}_j[x_k]\}_{j\neq i})$, and then the functionality sets $\mathsf{K}_j[x_k] := \mathsf{M}_j[x_k] \oplus x_k\Delta_j$ for each $j \neq i$.

– If P_j is corrupted where $j \neq i$, P_j gets to choose $\mathsf{K}_j[x_k]$, and then the functionality defines $\mathsf{M}_j[x_k] := \mathsf{K}_j[x_k] \oplus x_k\Delta_j$.

Leakage of Global Keys: If P_i is corrupted, the adversary may input $(\mathsf{leak}, j, S, \{\Delta'[k]\}_{k\in S})$. If P_j is honest, the functionality does the following:

– If there exists some $k \in S$ such that $\Delta'[k] \neq \Delta_j[k]$, the functionality outputs `fail` to all parties and aborts.

– Otherwise, it outputs `success` to the adversary and proceeds as if nothing has happened.

Figure 3: Functionality for multi-party authenticated bits with weak global keys

generality, we fix an honest party P_j to analyze the correlation check. When acting as a receiver in the extend command of $\mathcal{F}_{\mathsf{COTe}}$, a corrupt party P_i may send a vector $\mathbf{x}^j_k$ for $k \in [\ell']$ to $\mathcal{F}_{\mathsf{COTe}}$, and receives an MAC $\mathsf{M}'_j[x^j_k] := \mathsf{K}_j[x^j_k] + \mathbf{x}^j_k * \Delta_j$ for $k \in [\ell']$. We take P_i's inputs $\mathbf{x}^j_1, \ldots, \mathbf{x}^j_{\ell'} \in \mathbb{F}_2^\kappa$ to be the rows of a $\ell' \times \kappa$ matrix. Let $\mathbf{x}^1_j, \ldots, \mathbf{x}^\kappa_j \in \mathbb{F}_2^{\ell'}$ be the columns of the same matrix. If P_i is semi-honest, then $\mathbf{x}^j_k$ for $k \in [\ell']$ is monochrome, and $\mathbf{x}^1_j, \ldots, \mathbf{x}^\kappa_j$ are all equal.

Lemma 1 ([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 $\mathbf{s} \in \{\mathbf{x}^l_j\}_{l\in[\kappa]}$, let $H_\mathbf{s} = \{l \in [\kappa] \mid \mathbf{s} = \mathbf{x}^l_j\}$. Then one of the following holds:

– *For all $l \in H_\mathbf{s}$ and any $\Delta^{(1)}_j, \Delta^{(2)}_j \in S_{\Delta_j}$, $\Delta^{(1)}_j[l] = \Delta^{(2)}_j[l]$.*

– *$|H_\mathbf{s}| \geq d_j$ and $|\{\Delta_j[H_\mathbf{s}]\}_{\Delta_j \in S_{\Delta_j}}| = 2^{d_j}$, 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 meaning of 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 1 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. In the first case, the bits of Δ_j are revealed by

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 each $j \neq i$, P_i acting as a receiver sends (init) to $\mathcal{F}_{\mathsf{COTe}}$, and P_j acting as a sender sends $(\mathsf{init}, \Delta_j)$ to $\mathcal{F}_{\mathsf{COTe}}$.

Generate Authenticated Bits: The parties execute 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 $\mathcal{F}_{\mathsf{COTe}}$, where P_j sends (extend, ℓ') to $\mathcal{F}_{\mathsf{COTe}}$ and P_i sends $(\mathsf{extend}, \ell', \mathbf{x}_1, \ldots, \mathbf{x}_{\ell'})$ to $\mathcal{F}_{\mathsf{COTe}}$. From the functionality, P_i gets the MACs $\{\mathsf{M}_j[x_k]\}_{k\in[\ell']}$, and P_j gets the local keys $\{\mathsf{K}_j[x_k]\}_{k\in[\ell']}$.

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

5. The parties call $\mathcal{F}_{\mathsf{Rand}}$ to obtain ℓ' random field elements $\chi_1, \ldots, \chi_{\ell'} \in \mathbb{F}_{2^\kappa}$.
6. P_i locally computes over $\mathbb{F}_{2^\kappa}$ the following values:

$$\mathbf{y}^i := \textstyle\sum_{k=1}^{\ell'} \chi_k \cdot x_k, \qquad \mathsf{M}_j[\mathbf{y}^i] := \textstyle\sum_{k=1}^{\ell'} \chi_k \cdot \mathsf{M}_j[x_k] \text{ for each } j \neq i$$

   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 any check fails, P_j aborts.

Figure 4: Protocol for generating multi-party authenticated bits with weak global keys

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 of $\hat{\mathbf{s}}$. Then, for $j \in \mathcal{H}, k \in [\ell']$, we can write the MAC (with an error) received by a malicious 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 $\mathbf{s} \in \{\mathbf{x}_j^1, \ldots, \mathbf{x}_j^\kappa\}$. 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 1 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 1, 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 $\mathsf{M}_j[x_k^j] = \mathsf{K}_j[x_k^j] + x_k^j \cdot \Delta_j$.

Analysis of Consistency Check: When acting as a receiver in the extend command of $\mathcal{F}_{\mathsf{COTe}}$, a malicious 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 $k \in [\ell']$,

[2]One can easily prove if there are two different $\mathbf{s}$ satisfying the second case of Lemma 1, then the correlation check will *not* pass.

x^j_k can be denoted as $x^j_k = x_k + \delta^j_k$, where $\delta^{j_0}_k = 0$. Based on Lemma 1, we prove that a malicious P_i cannot use inconsistent values x^j_k to different honest parties.

Lemma 2. *For a corrupt party P_i and every honest party $P_j \in \mathcal{H}$, 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 $j \in \mathcal{H}$, $\delta^j_k = 0$.*

Proof. For each $j \in \mathcal{H}$, we define P_i's MAC on $\sum_{k=1}^{\ell'} \chi_k \cdot x^j_k$ as $\mathsf{M}_j[\mathbf{y}^i] = \sum_{k=1}^{\ell'} \chi_k \cdot \mathsf{M}_j[x^j_k]$, and P_j's local key on the same value as $\mathsf{K}_j[\mathbf{y}^i] = \sum_{k=1}^{\ell'} \chi_k \cdot \mathsf{K}_j[x^j_k]$. According to Lemma 1 and the above related analysis, for each $j \in \mathcal{H}, k \in [\ell']$, $\mathsf{M}_j[x^j_k] = \mathsf{K}_j[x^j_k] + x^j_k \cdot \Delta_j$ is known by the adversary who corrupts P_i . In Step 6 of protocol Π_{aBit}, P_i may broadcast an incorrect value $\hat{\mathbf{y}}^i = \mathbf{y}^i + \mathbf{e}^i$ 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 $P_j \in \mathcal{H}$ does not abort, then $\widehat{\mathsf{M}}_j[\mathbf{y}^i] = \mathsf{K}_j[\mathbf{y}^i] + \hat{\mathbf{y}}^i \cdot \Delta_j$. Thus, we have:

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

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

1. If $\mathsf{E}_{i,j} \neq \mathbf{0}$, then $(\mathbf{e}^i + \sum_{k=1}^{\ell'} \chi_k \cdot \delta^j_k) \neq \mathbf{0}$, and thus the adversary corrupting P_i 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. If $\mathsf{E}_{i,j} = \mathbf{0}$, then $\mathbf{e}^i = \sum_{k=1}^{\ell'} \chi_k \cdot \delta^j_k$ unless $\Delta_j = \mathbf{0}$ with probability $2^{-\kappa}$. As $\delta^{j_0}_k = 0$ for each $k \in [\ell']$, this implies that $\mathbf{e}^i = \mathbf{0}$. Thus, for each $j \in \mathcal{H}\backslash\{j^0\}$, we have that $\sum_{k=1}^{\ell'} \chi_k \cdot \delta^j_k = \mathbf{0}$. This happens with probability $2^{-\kappa}$, as $\{\delta^j_k\}_{k\in[\ell']}$ are independent of $\{\chi_k\}_{k\in[\ell']}$ and $\chi_1, \ldots, \chi_{\ell'}$ are uniformly random.

□

In addition, we will use the following lemma.

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

Based on the above three lemmas, we prove the following theorem.

Theorem 1. *Protocol Π_{aBit} shown in Figure 4 securely realizes the 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 $\mathcal{M} \in [n]$. Let $\mathcal{H}$ be the set of honest parties, i.e., $\mathcal{H} = [n]\backslash\mathcal{M}$. We construct a PPT simulator $\mathcal{S}$ that has access to the functionality $\mathcal{F}_{\mathsf{aBit}}$ and simulates $\mathcal{A}$'s view.

Case 1 ($P_i \in \mathcal{H}$)**:** $\mathcal{S}$ interacts with $\mathcal{A}$ and simulates as follows:

1. For each $P_j \in \mathcal{M}$, $\mathcal{S}$ 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 $j \in \mathcal{M}$, $\mathcal{S}$ computes $\mathsf{K}_j[\mathbf{y}^i] := \sum_{k=1}^{\ell'} \chi_k \cdot \mathsf{K}_j[x_k]$, picks random $\mathbf{y}^i \leftarrow \mathbb{F}_{2^\kappa}$, 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 $j \in \mathcal{H} \backslash \{i\}$, $\mathcal{S}$ samples a random $\mathsf{M}_j[\mathbf{y}^i] \leftarrow \mathbb{F}_{2^\kappa}$ and sends it to dummy party P_j.

Case 2 ($P_i \in \mathcal{M}$): $\mathcal{S}$ interacts with $\mathcal{A}$ and simulates as follows:

1. For each corrupt party $P_j \in \mathcal{M} \backslash \{P_i\}$, $\mathcal{S}$ receives Δ_j from $\mathcal{A}$ for $\mathcal{F}_{\mathsf{COTe}}$, and then sends Δ_j to $\mathcal{F}_{\mathsf{aBit}}$. For each $j \in \mathcal{H}$, $\mathcal{S}$ emulates the functionality $\mathcal{F}_{\mathsf{COTe}}$ 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 P_i.

2. For each $j \in \mathcal{H}$, let $\hat{\mathsf{s}}$ and $H_{\hat{\mathsf{s}}}$ be as in Lemma 1, i.e., $|H_{\hat{\mathsf{s}}}| \geq d$ and $\mathbf{x}_j^l = \mathbf{x}_j^{l'}$ for all $l, l' \in H_{\hat{\mathsf{s}}}$. This implies $\mathbf{x}_k^j[l] = \mathbf{x}_k^j[l']$ for all $l, l' \in H_{\hat{\mathsf{s}}}$ and $k \in [\ell']$. For each $k \in [\ell']$, $\mathcal{S}$ sets $x_k^j := \mathbf{x}_k^j[l]$ for some $l \in H_{\hat{\mathsf{s}}}$. For each $j \in \mathcal{H}$, $\mathcal{S}$ 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 $c_j := |S_j|$.

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 elements to $\mathcal{A}$.

4. For each $j \in \mathcal{H}$, $\mathcal{S}$ computes $\mathbf{y}^i := \sum_{k=1}^{\ell'} \chi_k \cdot x_k^j$, and receives $\hat{\mathbf{y}}^i$ from $\mathcal{A}$ over a broadcast channel. Then, $\mathcal{S}$ computes $\mathbf{e}^{i,j} := \hat{\mathbf{y}}^i + \mathbf{y}^i$. If $\mathbf{e}^{i,j} \neq \mathbf{0}$, $\mathcal{S}$ 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 $S_j = \emptyset$, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and halts if $E_j \neq \mathbf{0}$, and sets $\mathbf{e}_k^j * \Delta_j = \mathbf{0}$ for $k \in [\ell']$ otherwise. Otherwise, $\mathcal{S}$ writes $E_j \in \mathbb{F}_{2^\kappa}$ as a vector in $\mathbb{F}_2^\kappa$.

5. For each $j \in \mathcal{H}$, 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 $\mathbf{X}_j$ is a random $\kappa \times |S_j|$ matrix determined by $\{\mathbf{e}_k^j\}_{k \in [\ell']}$ and $\{\chi_k\}_{k \in [\ell']}$, and $\mathbf{t}_j$ is an unknown 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, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and halts.
   - If there is an unique solution for the equation (i.e., $\mathbf{X}_j$ has rank c_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 $j \in \mathcal{H}$, 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 `fail` from $\mathcal{F}_{\mathsf{aBit}}$, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and halts. Otherwise, $\mathcal{S}$ receives `success` from $\mathcal{F}_{\mathsf{aBit}}$ and is confirmed $\Delta_j[l] = \Delta'_j[l]$ for each $l \in S_j$.

7. For each $j \in \mathcal{H}$, 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 $l \in H_{\hat{\mathsf{s}}}$, 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 $j, j' \in \mathcal{H}$ 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 $j \in \mathcal{H}$.

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

In both cases, if $\mathcal{S}$ does not abort, $\mathcal{S}$ outputs what $\mathcal{A}$ outputs.

Analysis of Simulation for Case 1. It is easy to see that the correlation and consistency checks pass. For $j \in \mathcal{H}\backslash\{i\}$, $\mathsf{M}_j[\mathbf{y}^i]$ sampled by $\mathcal{S}$ has the same distribution as the one in the real-world 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 $\mathbb{F}_{2^\kappa}$ and kept secret. Below, all we need to do is to prove that $\mathbf{y}^i$ and $\mathsf{M}_j[\mathbf{y}^i]$ for $j \in \mathcal{M}$ sent by $\mathcal{S}$ are statistically indistinguishable from the values sent by P_i in the real-world protocol execution.

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

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

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 $E_j = \sum_{k=1}^{\ell'} \chi_k \cdot \big(\mathbf{e}_k^j * \Delta_j\big) + \mathbf{e}^{i,j} \cdot \Delta_j$. Since $\mathcal{A}$ knows $\mathbf{e}_k^j * \Delta_j$ for $k \in [\ell']$ and E_j, we have that $\mathbf{e}^{i,j} = \mathbf{0}$ 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 1, we have that S_{Δ_j} restricted to the bits corresponding to $H_{\hat{\mathbf{s}}}$ has entropy at least d_j. Therefore, with probability at most 2^{-d_j}, $\mathcal{A}$ guesses successfully the bits $\{\Delta_j[l]\}_{l \in H_{\hat{\mathbf{s}}}}$. Overall, the probability that $\mathbf{e}^{i,j} \neq \mathbf{0}$ 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 $2^{-\kappa} + 2^{-(\kappa+d_j)}$, which is negligible in κ. As a result, if the protocol does not abort, we have that $E_j = \sum_{k=1}^{\ell'} \chi_k \cdot \big(\mathbf{e}_k^j * \Delta_j\big)$ in both worlds with overwhelming probability.

If $S_j = \emptyset$, i.e., $\mathbf{e}_k^j = \mathbf{0}$ for all $k \in [\ell']$, then it is easy to see that the simulation of $\mathcal{S}$ is statically indistinguishable from the real protocol execution. Below, for each $j \in \mathcal{H}$, 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 an 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. Therefore, in the third case, the simulation of $\mathcal{S}$ is statistically indistinguishable from the real protocol execution.

From Lemma 2, the probability that $\mathcal{S}$ aborts in Step 8 of the simulation is negligible in κ. In all, we complete the proof. □

3.2 Improved Multi-Party Authenticated Shares

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

Running with parties $P_1, \dots, P_n$, the functionality operates as follows:

Honest Parties:

- Upon receiving (init) from all parties, for each $i \in [n]$, sample $\Delta_i \leftarrow \{0,1\}^\kappa$ and send it to P_i.
- Upon receiving (aShare, ℓ) from all parties, sample $x_1, \dots, x_\ell \leftarrow \{0,1\}$ and generate random authenticated shares $\{\langle x_k \rangle = ([x_k^1]_1, \dots, [x_k^n]_n)\}_{k \in [\ell]}$, and then send them to the parties.
  In detail, for each $k \in [\ell]$, pick $x_k^1, \dots, x_k^n \leftarrow \{0,1\}$ such that $\bigoplus_{i\in[n]} x_k^i = x_k$. For each $i \in [n], k \in [\ell]$, sample $\mathsf{K}_j[x_k^i] \leftarrow \{0,1\}^\kappa$ and compute $\mathsf{M}_j[x_k^i] := \mathsf{K}_j[x_k^i] \oplus x_k^i \Delta_j$ for $j \neq i$. For each $i \in [n]$, send $(x_k^i, \{\mathsf{M}_j[x_k^i], \mathsf{K}_i[x_k^j]\}_{j\neq i})$ to P_i.

Corrupt Parties: Corrupt parties can choose their own randomness used to compute the outputs they receive from the functionality.

Leakage of global keys: The adversary may input $(\mathsf{leak}, i, S, \{\Delta'[k]\}_{k\in S})$. If P_i is honest, the functionality does the following:

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

Figure 5: Functionality for multi-party authenticated shares with weak global keys

We propose an efficient protocol Π_{aShare}, which allows n parties to generate authenticated shares of a secret bit. 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 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 details of our protocol Π_{aShare} are described in Figure 6. We know that $\mathbb{F}_{2^\kappa} \cong \mathbb{F}_2[X]/(X^\kappa + X + 1)$. Let $\mathsf{X} \in \mathbb{F}_{2^\kappa}$ be a root of the irreducible polynomial $X^\kappa + X + 1 \in \mathbb{F}_2[X]$. If every party P_i picks $r_1^i, \dots, r_\kappa^i \leftarrow \{0,1\}$ at random, then $\mathbf{y}^i = \sum_{h=1}^{\kappa} \mathsf{X}^{h-1} \cdot r_h^i$ is uniformly random over $\mathbb{F}_{2^\kappa}$. Protocol Π_{aShare} aims to securely realize a functionality $\mathcal{F}_{\mathsf{aShare}}$ for authenticated shares with weak global keys as shown in Figure 5. The detailed security proof can be found in Section B.

Implementing the aBit command of $\mathcal{F}_{\mathsf{prep}}$. In the preprocessing 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 invoke the protocol Π_{aBit} described in Figure 4 by n times to implement the aBit command of $\mathcal{F}_{\mathsf{prep}}$. In this case, a malicious party P_i may use inconsistent global keys in n different

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 by calling authenticated bit functionality $\mathcal{F}_{\mathsf{aBit}}$.

3. For each $i \in [n]$, all parties send $(\mathsf{aBit}, i, \ell + \kappa)$ to $\mathcal{F}_{\mathsf{aBit}}$, which samples $x_1^i, \ldots, x_\ell^i, r_1^i, \ldots, r_\kappa^i \leftarrow \{0,1\}$ and sends multi-party random authenticated bits $\left\{[x_k^i]_i\right\}_{k\in[\ell]}$ and $\left\{[r_h^i]_i\right\}_{h\in[\kappa]}$ to the parties.

Check Consistency: The parties check the consistency of global keys as follows:

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

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

5. Every party P_i obtains a random zero-share $\mathbf{u}^i \in \mathbb{F}_{2^\kappa}$ such that $\sum_{i=1}^n \mathbf{u}^i = \mathbf{0}$ 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

$$\left(\left\{\mathbf{z}_j^i := \mathsf{M}_j[\mathbf{y}^i]\right\}_{j\neq i}, \mathbf{z}_i^i\right) \in \mathbb{F}_{2^\kappa}^n$$

   by the Commit command of $\mathcal{F}_{\mathsf{Com}}$.
8. After all commitments have been made, all parties open their commitments by the Open command of $\mathcal{F}_{\mathsf{Com}}$, and then check the following holds:

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

   If the check fails, abort.

Figure 6: Protocol for generating multi-party authenticated shares with weak global keys

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 the 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 shown in protocol Π_{aShare} so as to guarantee the consistency of global keys.

4 Improved Multi-Party Authenticated Garbling in $\mathcal{F}_{\mathsf{prep}}$

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 7 and Figure 8. In protocol Π_{mpc}, we use an amortized opening process for authenticated bits/shares described in Section A.4, which has 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.

We outline each step in the protocol Π_{mpc} as follows:

1. In the function-independent phase, each party P_i obtains its own private global key Δ_i from $\mathcal{F}_{\mathsf{prep}}$ (Step 1). All parties generate authenticated bits, shares and AND triples via calling $\mathcal{F}_{\mathsf{prep}}$ (Steps 2-4). Each garbler P_i $(i \neq 1)$ samples random garbled labels on 0-bit for all circuit-input wires (Step 5).

2. Using the authenticated Beaver triple technique [Bea92, BDOZ11], the parties can compute the authenticated share of the product of two wire masks using a random authenticated AND triple (Step 6-7). Based on the authenticated shares, all parties compute a distributed garbled circuit (Step 8).

3. Every party masks their input bits with the wire masks, and then broadcasts the public values to all other parties (Step 9(a)). Evaluator P_1 also receives the garbled labels for all circuit-input wires from each garbler (Step 9(b)).

4. P_1 locally evaluates the circuit following the topological order (Step 10). Then, P_1 checks the correctness of public values on the output wires of all AND gates (Steps 11-12).

5. For each circuit-output wire associated with P_i's output, all parties reveal the wire mask to P_i, who can unmask the public value and obtain its output bit (Step 13).

Note that if there are two parties that obtain the same output such as P_i and P_j (i.e., $\mathcal{O}_i = \mathcal{O}_j$), 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 Section D, we give a detailed security proof of protocol Π_{mpc}. In particular, we are able to prove the following result.

Theorem 2. *Let $f : \{0,1\}^{n|\mathcal{I}|} \to \{0,1\}^{n|\mathcal{O}|}$ be an n-party functionality. Then the protocol Π_{mpc} shown in Figures 7 and 8 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.*

5 Performance Evaluation

In this section, we discuss the performance improvement of our MPC protocol compared to state-of-the-art constant-round maliciously secure 2PC protocol [KRRW18] (KRRW) and MPC protocol [WRK17b] (WRK) tolerating arbitrary number of corruptions. Hazay, Ishai and Venkitasubramaniam [HIV17] proposed constant-overhead protocols for maliciously secure two-party computation. However, since their protocol with full security is not concretely efficient, we do not compare with it. Hazay, Scholl and Soria-Vazquez (HSS) [HSS17] also proposed a constant-round MPC protocol with performance mostly similar to WRK, and thus our comparison to WRK also applies to HSS. For all protocols, we choose the computational security parameter $\kappa = 128$ and statistical security parameter $\rho = 40$. We also include some discussion when $\rho = 64$ in Section E of the Supplementary Material, where we are able to observe a better improvement. We didn't implement our protocol; all numbers below are either obtained from prior work or calculated given the protocol description.

Protocol Π_{mpc}, preprocessing phases

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 associated with a wire $w \in \mathcal{I}_i$.

Function-independent phase:

1. All parties send (init) to $\mathcal{F}_{\mathsf{prep}}$, which sends a random $\Delta_i \in \{0,1\}^\kappa$ to P_i for each $i \in [n]$ such that $\mathsf{lsb}(\Delta_i) = 1$ if $i = 2$.
2. For each $i \in [n]$ and $w \in \mathcal{I}_i$, the parties send (aBit, i) to $\mathcal{F}_{\mathsf{prep}}$, which samples a random authenticated bit $[\lambda_w]_i$, and sends $(\lambda_w, \{\mathsf{M}_j[\lambda_w]\}_{j\neq i})$ to P_i and $\mathsf{K}_j[\lambda_w]$ to P_j for each $j \neq i$. Then the parties define an authenticated share $\langle \lambda_w \rangle$ via running $\mathsf{Bit2Share}([\lambda_w]_i)$.
3. For each wire $w \in \mathcal{W}$, the parties send (aShare) to $\mathcal{F}_{\mathsf{prep}}$, which generates a random authenticated share $\langle \lambda_w \rangle = ([\lambda^1_w]_1, \ldots, [\lambda^n_w]_n)$, and sends $(\lambda^i_w, \{\mathsf{M}_j[\lambda^i_w], \mathsf{K}_i[\lambda^j_w]\}_{j\neq i})$ to P_i for each $i \in [n]$.
4. For each wire $w \in \mathcal{W}$, the parties send (aAND) to $\mathcal{F}_{\mathsf{prep}}$, which sends a random authenticated AND triple $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ to the parties.
5. For each circuit-input wire $w \in \mathcal{I}_1 \cup \cdots \cup \mathcal{I}_n$, P_i picks $\mathsf{L}^i_{w,0} \leftarrow \{0,1\}^\kappa$ for $i \neq 1$.

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 every $i \neq 1$, 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 the following in parallel:
   (a) Take a fresh unused 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 every AND gate $(\alpha, \beta, \gamma, \wedge)$, for each $i \neq 1$, P_i computes $\mathsf{L}^i_{\alpha,1} := \mathsf{L}^i_{\alpha,0} \oplus \Delta_i$ and $\mathsf{L}^i_{\beta,1} := \mathsf{L}^i_{\beta,0} \oplus \Delta_i$, and then computes the following values:

$$\begin{aligned}
\mathcal{G}^i_{\gamma,0} &:= \mathsf{H}(\mathsf{L}^i_{\alpha,0}, \gamma, 0) \oplus \mathsf{H}(\mathsf{L}^i_{\alpha,1}, \gamma, 0) \oplus (\textstyle\bigoplus_{j\neq i} \mathsf{K}_i[\lambda^j_\beta]) \oplus \lambda^i_\beta \Delta_i \\
\mathcal{G}^i_{\gamma,1} &:= \mathsf{H}(\mathsf{L}^i_{\beta,0}, \gamma, 1) \oplus \mathsf{H}(\mathsf{L}^i_{\beta,1}, \gamma, 1) \oplus \mathsf{L}^i_{\alpha,0} \oplus (\textstyle\bigoplus_{j\neq i} \mathsf{K}_i[\lambda^j_\alpha]) \oplus \lambda^i_\alpha \Delta_i \\
\mathsf{L}^i_{\gamma,0} &:= \mathsf{H}(\mathsf{L}^i_{\alpha,0}, \gamma, 0) \oplus \mathsf{H}(\mathsf{L}^i_{\beta,0}, \gamma, 1) \oplus (\textstyle\bigoplus_{j\neq i} \mathsf{K}_i[\lambda^j_{\alpha\beta}]) \oplus \lambda^i_{\alpha\beta} \Delta_i \oplus (\textstyle\bigoplus_{j\neq i} \mathsf{K}_i[\lambda^j_\gamma]) \oplus \lambda^i_\gamma \Delta_i \\
b_\gamma &:= \mathsf{lsb}(\mathsf{L}^i_{\gamma,0}) \text{ if } i = 2 \\
\text{For} \quad & u, v \in \{0,1\},\ \left\{\mathsf{M}_j[r^i_{uv}] := u \cdot \mathsf{M}_j[\lambda^i_\beta] \oplus v \cdot \mathsf{M}_j[\lambda^i_\alpha] \oplus \mathsf{M}_j[\lambda^i_{\alpha\beta}] \oplus \mathsf{M}_j[\lambda^i_\gamma]\right\}_{j\neq i,1} \\
G^{i,j}_{\gamma,00} &:= \mathsf{H}(\mathsf{L}^i_{\alpha,0}, \mathsf{L}^i_{\beta,0}, \gamma, j) \oplus \mathsf{M}_j[r^i_{00}] \text{ for each } j \neq i, 1 \\
G^{i,j}_{\gamma,01} &:= \mathsf{H}(\mathsf{L}^i_{\alpha,0}, \mathsf{L}^i_{\beta,1}, \gamma, j) \oplus \mathsf{M}_j[r^i_{01}] \text{ for each } j \neq i, 1 \\
G^{i,j}_{\gamma,10} &:= \mathsf{H}(\mathsf{L}^i_{\alpha,1}, \mathsf{L}^i_{\beta,0}, \gamma, j) \oplus \mathsf{M}_j[r^i_{10}] \text{ for each } j \neq i, 1 \\
G^{i,j}_{\gamma,11} &:= \mathsf{H}(\mathsf{L}^i_{\alpha,1}, \mathsf{L}^i_{\beta,1}, \gamma, j) \oplus \mathsf{M}_j[r^i_{11}] \text{ for each } j \neq i, 1
\end{aligned}$$

For each $w \in \mathcal{W}$, 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 7: The preprocessing phases of our MPC protocol in the $\mathcal{F}_{\mathsf{prep}}$-hybrid model

Protocol Π_{mpc}, online phase

Input processing:

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 $\mathsf{L}^j_{w,\Lambda_w} := \mathsf{L}^j_{w,0} \oplus \Lambda_w \Delta_j$ to P_1.

Circuit evaluation:

10. P_1 evaluates the circuit following the topological order. For each gate $(\alpha, \beta, \gamma, T)$, P_1 holds $(\Lambda_\alpha, \{\mathsf{L}^i_{\alpha,\Lambda_\alpha}\}_{i\neq 1})$ and $(\Lambda_\beta, \{\mathsf{L}^i_{\beta,\Lambda_\beta}\}_{i\neq 1})$, and computes as follows:
    – If $T = \oplus$, compute $\Lambda_\gamma := \Lambda_\alpha \oplus \Lambda_\beta$ and $\left\{\mathsf{L}^i_{\gamma,\Lambda_\gamma} := \mathsf{L}^i_{\alpha,\Lambda_\alpha} \oplus \mathsf{L}^i_{\beta,\Lambda_\beta}\right\}_{i\neq 1}$.
    – If $T = \wedge$, let $u = \Lambda_\alpha$ and $v = \Lambda_\beta$, and compute the following:
    (a) For each $j \neq 1$, $\mathsf{M}_j[r^1_{uv}] := \Lambda_\alpha \cdot \mathsf{M}_j[\lambda^1_\beta] \oplus \Lambda_\beta \cdot \mathsf{M}_j[\lambda^1_\alpha] \oplus \mathsf{M}_j[\lambda^1_{\alpha\beta}] \oplus \mathsf{M}_j[\lambda^1_\gamma]$.
    (b) For $i \neq 1$ and $j \neq i, 1$, $\mathsf{M}_j[r^i_{uv}] := \mathsf{H}(\mathsf{L}^i_{\alpha,\Lambda_\alpha}, \mathsf{L}^i_{\beta,\Lambda_\beta}, \gamma, j) \oplus G^{i,j}_{\gamma,uv}$.
    (c) For each $i \neq 1$, compute the garbled label on the output wire
    $$\mathsf{L}^i_{\gamma,\Lambda_\gamma} := \mathsf{H}(\mathsf{L}^i_{\alpha,\Lambda_\alpha}, \gamma, 0) \oplus \mathsf{H}(\mathsf{L}^i_{\beta,\Lambda_\beta}, \gamma, 1) \oplus \Lambda_\alpha \mathcal{G}^i_{\gamma,0} \oplus \Lambda_\beta(\mathcal{G}^i_{\gamma,1} \oplus \mathsf{L}^i_{\alpha,\Lambda_\alpha}) \oplus (\textstyle\bigoplus_{j\neq i} \mathsf{M}_i[r^j_{uv}]).$$
    (d) Compute the public value $\Lambda_\gamma := b_\gamma \oplus \mathsf{lsb}(\mathsf{L}^2_{\gamma,\Lambda_\gamma})$.

Check public values:

11. P_1 computes $h_i := \mathsf{H}(\{\mathsf{L}^i_{w,\Lambda_w}\}_{w\in\mathcal{W}})$ for each $i \neq 1$, and samples a seed $\chi \leftarrow \mathbb{F}_{2^\kappa}$. For each $i \neq 1$, P_1 sends $\{\Lambda_w\}_{w\in\mathcal{W}}$ and h_i along with χ to P_i. Then, P_i checks that $h_i = \mathsf{H}(\{\mathsf{L}^i_{w,0} \oplus \Lambda_w \Delta_i\}_{w\in\mathcal{W}})$. 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 that $t_\gamma = (\Lambda_\alpha \oplus \lambda_\alpha) \wedge (\Lambda_\beta \oplus \lambda_\beta) \oplus (\Lambda_\gamma \oplus \lambda_\gamma) = 0$ 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
    $$\mathsf{M}_1[t^i_\gamma] := \Lambda_\alpha \cdot \mathsf{M}_1[\lambda^i_\beta] \oplus \Lambda_\beta \cdot \mathsf{M}_1[\lambda^i_\alpha] \oplus \mathsf{M}_1[\lambda^i_{\alpha\beta}] \oplus \mathsf{M}_1[\lambda^i_\gamma].$$
    (b) For each AND gate $(\alpha, \beta, \gamma, \wedge)$, P_1 computes the following values:
    $$\begin{aligned} &t^1_\gamma := \Lambda_\alpha \cdot \Lambda_\beta \oplus \Lambda_\gamma \oplus \Lambda_\alpha \cdot \lambda^1_\beta \oplus \Lambda_\beta \cdot \lambda^1_\alpha \oplus \lambda^1_{\alpha\beta} \oplus \lambda^1_\gamma \\ &\mathsf{K}_1[t^i_\gamma] := \Lambda_\alpha \cdot \mathsf{K}_1[\lambda^i_\beta] \oplus \Lambda_\beta \cdot \mathsf{K}_1[\lambda^i_\alpha] \oplus \mathsf{K}_1[\lambda^i_{\alpha\beta}] \oplus \mathsf{K}_1[\lambda^i_\gamma] \text{ for each } i \neq 1 \\ &\mathsf{M}_1[t^1_\gamma] := (\textstyle\bigoplus_{i\neq 1} \mathsf{K}_1[t^i_\gamma]) \oplus t^1_\gamma \Delta_1 \end{aligned}$$
    (c) Let $\mathbf{H}$ be an almost universal linear hash function defined by χ. For each $i \neq 1$, P_i computes $\mathbf{c}_i := \mathbf{H}(\{\mathsf{M}_1[t^i_w]\}_{w\in\mathcal{W}}) \in \mathbb{F}_2^\kappa$, and then sends $\mathbf{c}_i$ to P_1.
    (d) P_1 computes $\mathbf{c}_1 := \mathbf{H}(\{\mathsf{M}_1[t^1_w]\}_{w\in\mathcal{W}}) \in \mathbb{F}_2^\kappa$, and then checks $\sum_{i=1}^n \mathbf{c}_i = \mathbf{0}$. If the check fails, P_1 aborts.

Output processing:

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 $\lambda^j_w := \mathsf{Open}([\lambda^j_w]^i_j)$.
    (b) P_i computes $y^i_w := \Lambda_w \oplus (\bigoplus_{j\in[n]} \lambda^j_w)$.

Figure 8: The online phase of our MPC protocol

#Parties	Protocol	Ind. (MB)		Dep.	Online	Total (MB)	
		$\#1$	$\#1024$	(MB)	(KB)	$\#1$	$\#1024$
$n = 2$	KRRW [KRRW18]	2.5	1.9	0.2	5.0	2.7	2.1
	Ours	1.9	1.4	0.2	4.2	2.1	1.6
$n = 3$	WRK [WRK17b]	4.8	3.6	1.0	6.3	5.8	4.6
	Ours	3.7	2.8	0.66	6.2	4.4	3.5
$n = 5$	WRK [WRK17b]	9.7	7.2	1.9	10.4	11.6	9.1
	Ours	7.5	5.7	1.5	10.3	9.0	7.2

Table 1: Comparison between our MPC protocol and the best known protocol in terms of communication overhead for secure AES evaluation. The columns for "$\#1$" and "$\#1024$" denote the communication cost over a single execution and the amortized communication cost over 1024 executions respectively.

Comparing communication based on AES circuit. An AES circuit consists of 6800 AND gates and 128 bits of input. 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 1 reports communication complexity for secure AES evaluation, where all numbers are the maximum amount of data sent by any one party per execution. We do not consider the cost of base OTs as it is very small and the same for all protocols discussed here. To make the comparison fair, we incorporate the optimization to reduce the size of the MACs (authenticated to the evaluator) from κ bits to ρ bits [WRK17a] in the multi-party setting. This reduces the size of the distributed garbled circuit in WRK 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 optimizing the online phase of their protocol using the amortized opening of authenticated bits. From the description of the online phase of the WRK protocol, the original online communication cost is much more than the one shown in the Table 1.

For a single execution, our protocol leads to an $1.3\times$ improvement in the function-independent phase, compared to KRRW and WRK. In terms of the amortized communication cost of 1024 executions, our protocol provides $1.26\times \sim 1.36\times$ improvement in the function-independent phase. Compared to WRK, our protocol gives about $1.52\times$ improvement for three-party case and $1.27\times$ improvement for five-party case in the communication of the function-dependent phase. While reducing the communication in both preprocessing phases, we do not increase the communication cost in the online phase. Overall, our protocol results in $\approx 1.3\times$ improvement for both a single execution and 1024 executions.

#Parties	Protocol	Hamming Distance			Sorting		
		Ind.	Dep.	Online	Ind.	Dep.	Online
$n = 2$	KRRW [KRRW18]	586.4	67.9	67.9	2649.4	327.8	5.5
	Ours	454.3	67.9	67.4	2134.9	327.8	4.2
$n = 3$	WRK [WRK17b]	1352.2	311.4	101.5	5319.7	1518.1	6.4
	Ours	942.2	202.6	100.9	4269.8	987.8	6.3
$n = 5$	WRK [WRK17b]	3056.6	580.9	169.1	10661.5	2831.8	10.6
	Ours	1884.3	472.1	168.0	8539.5	2301.5	10.5

Table 2: Communication cost of our and prior best protocols for computing Hamming distance and sorting. All numbers are in megabytes (MB) with a single execution.

Comparing communication based on other circuits. In Table 2, we also compare our protocol with the best known protocols 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 bit-strings. In the two-party setting, every party directly inputs a bit string of length 1048576 bits rather than XOR-share.

– **Sorting.** Each party inputs an XOR-share of 4096 32-bit numbers. The circuit first XORs them to recover the underlying numbers, and then sorts the numbers.

Here, we only compare the 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 $1.3\times$ improvement for secure two-party computation, and $1.44\times$ and $1.62\times$ improvements for three-party and five-party cases respectively. For sorting circuit, our protocol gives a $1.24\times \sim 1.25\times$ 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 KRRW (resp., WRK), our protocol reduces the total communication by more than 130 MB (resp., 500 MB when $n = 3$ and 1 GB if $n = 5$) for hamming distance and 500 MB (resp., 1.5 GB when $n = 3$ and 2.5 GB if $n = 5$) for sorting.

References

[AOR+19] Abdelrahaman Aly, Emmanuela Orsini, Dragos Rotaru, Nigel P. Smart, and Tim Wood. Zaphod: Efficiently combing lsss and garbled circuits in scale. Cryptology ePrint Archive, Report 2019/974, 2019. https://eprint.iacr.org/2019/974.

[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.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conf. on Computer and Communications Security (CCS) 1993*, pages 62–73. ACM Press, 1993.

[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.

[DEF$^+$19] Ivan Damgård, Daniel Escudero, Tore 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*, 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.

[GKWY20] Chun Guo, Jonathan Katz, Xiao Wang, and Yu Yu. Efficient and secure multiparty computation from fixed-key block ciphers. In *IEEE Symposium on Security and Privacy (S&P) 2020*, 2020. Available at `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.

[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.

[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. Cryptology ePrint Archive, Report 2019/207, 2019. `https://eprint.iacr.org/2019/207`.

[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 Communication Model

We assume that all parties are connected via authenticated channels, as well as point-to-point channels and a broadcast channel. The default method of communication in our protocols is authenticated channel, unless otherwise specified. In practice, these channels can be implemented using standard techniques. In particular, a broadcast channel can be efficiently implemented using a standard 2-round echo-broadcast protocol [GL05], as we allow abort. When each party needs to broadcast m messages of ℓ-bit length, the communication overhead of this broadcast protocol can be improved from $nm\ell$ bits to $m\ell + \rho$ bits using the optimized technique [DPSZ12] and almost universal linear hash functions [CDD+16].

A.2 Commitment and Coin-tossing Functionalities

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

Commit phase: On input $(\mathsf{Commit}, i, x, \tau_x)$ from P_i, record (i, x, τ_x) and output $(\mathsf{Receipt}, i, \tau_x)$ to all parties. Ignore any future Commit messages with the same session identifier τ_x from P_i.

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

Figure 9: The functionality for commitments

Commitment. We will use a commitment functionality shown in Figure 9. This functionality can easily be implemented in the random oracle model [HSS17] via executing as follows:

Commit phase: A party P_i picks a random $r \leftarrow \{0,1\}^\kappa$ and computes $C_x^i = \mathsf{Commit}(i, x) := H(i, x, r)$, and then broadcasts C_x^i to all parties, where $H : \{0,1\}^* \to \{0,1\}^{2\kappa}$ is a hash function modeled as a random oracle [BR93].

Open phase: P_i sends (x, r) to all parties, who check if $C_x^i = H(i, x, r)$.

Coin tossing. We will use a standard coin-tossing functionality $\mathcal{F}_{\mathsf{Rand}}$ shown in Figure 10, which can be securely realized with commitments to random seeds. In particular, every party commits to a seed via $\mathcal{F}_{\mathsf{Com}}$, and then opens the seed and uses the XOR of the seeds from all parties to seed a pseudorandom generator PRG, which is secure in the random oracle model.

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

$\mathcal{F}_{\mathsf{Rand}}$ runs with parties $P_1, \ldots, P_n$ as follows:

– Upon receiving (Rand, ℓ) from all parties, sample $r \leftarrow \{0,1\}^\ell$, and then send r to all parties.

Figure 10: Coin-tossing functionality

A.3 Almost Universal Linear Hash Functions

We will use a family of almost universal linear hash functions [CDD+16] over $\mathbb{F}_2$ defined as follows:

Definition 1 (Almost Universal Linear Hashing). *We say that a family $\mathcal{H}$ of linear hash functions $\mathbb{F}_2^m \to \mathbb{F}_2^s$ is ϵ-almost universal, if it holds for every non-zero $\mathbf{x} \in \mathbb{F}_2^m$ such that*

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

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 $\chi \leftarrow \mathbb{F}_{2^s}$.
- Let $\ell = m/s$. Here we assume that s divides m for the sake of simplicity.
- Use χ to define the following linear hash function $\mathbf{H}$:

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

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 PCLMULQDQ [NST17]. This construction described as above provides an almost universal family with $\epsilon = \ell \cdot 2^{-s} = \frac{m}{s} \cdot 2^{-s}$, as χ is uniformly random in $\mathbb{F}_{2^s}$ and independent of the input $\mathbf{x} = (x_1, x_2, \ldots, x_\ell)$. This can be improved to 2^{-s}, at the cost of a larger seed, by using ℓ different elements $\chi_i \in \mathbb{F}_{2^s}$.

A.4 Amortized Opening of Authenticated Bits and Shares

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], the validity of multiple authenticated bits can be checked in an amortized way, i.e., it is possible to open ℓ authenticated bits with less than ℓ times the communication. Specifically, we have the following amortized opening process.

- 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 $x_1, \ldots, x_\ell$ along with $\tau_j := \mathsf{H}(\text{“open”}, \mathsf{M}_j[x_1], \ldots, \mathsf{M}_j[x_\ell])$ to P_j.
  2. P_j checks that $\tau_j = \mathsf{H}(\text{“open”}, \mathsf{K}_j[x_1] \oplus x_1\Delta_j, \ldots, \mathsf{K}_j[x_\ell] \oplus x_\ell\Delta_j)$. 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 $j \neq i$.

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

  1. For each $j \neq i$, P_i sends $x_1^i, \ldots, x_\ell^i$ along with $\tau_{i,j} := \mathsf{H}(\text{“open”}, \mathsf{M}_j[x_1^i], \ldots, \mathsf{M}_j[x_\ell^i])$ to P_j
  2. For each $j \neq i$, P_j checks that $\tau_{i,j} = \mathsf{H}(\text{“open”}, \mathsf{K}_j[x_1^i] \oplus x_1^i\Delta_j, \ldots, \mathsf{K}_j[x_\ell^i] \oplus x_\ell^i\Delta_j)$, 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 leaks 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 4. *If* H *is a random oracle, then the opening process in the above amortized way for two-party authenticated bits guarantees that 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.
The PPT adversary $\mathcal{A}$ *corrupting* P_i *is allowed to leak* c *bits of* Δ_j*, but* 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 $\tau'_j = \mathsf{H}(\text{"open"}, \mathsf{K}[x_1] \oplus x'_1\Delta_j, \ldots, \mathsf{K}[x_\ell] \oplus x'_\ell\Delta_j)$. If $\mathcal{A}$ makes a query $\mathbf{z}$ to H such that $\mathsf{H}(\mathbf{z}) = \tau'_j$ but $\mathbf{z} \neq (\text{"open"}, \mathsf{K}[x_1] \oplus x'_1\Delta_j, \ldots, \mathsf{K}[x_\ell] \oplus x'_\ell\Delta_j)$, 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 $1/2^{\kappa-c}$. 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 Security Proof of Our Authenticated Share Protocol

First we can see that Π_{aShare} is correct when all parties are honest, because

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

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 $R_{i,j_0} = \mathbf{0}$. 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 $P_j \in \mathcal{H}$.

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

Proof. In Step 6 of protocol Π_{aShare}, if all corrupt parties are semi-honest, then every party P_i broadcasts $\tilde{\mathbf{y}}^i$ and computes $\mathbf{y} := \sum_{i=1}^n \tilde{\mathbf{y}}^i$. However, every malicious party $P_i \in \mathcal{M}$ may broadcast an adversarial value $\hat{\mathbf{y}}^i$, such that $\hat{\mathbf{y}} := \sum_{i\in\mathcal{M}} \hat{\mathbf{y}}^i + \sum_{i\in\mathcal{H}} \tilde{\mathbf{y}}^i = \mathbf{y} + \mathbf{e}$, 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 $k \in \mathcal{M}$, in such a way that $\sum_{k\in\mathcal{M}} \hat{\mathbf{z}}_i^k = \sum_{k\in\mathcal{M}} \mathbf{z}_i^k + E_i$, 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 holds:

$$\begin{aligned}
\mathbf{0} &= \sum_{j\in\mathcal{H}} \mathbf{z}_i^j + \sum_{j\in\mathcal{M}} \hat{\mathbf{z}}_i^j = \mathbf{z}_i^i + \sum_{j\neq i} \mathbf{z}_i^j + E_i \\
&= \Big(\sum_{j\neq i} \mathsf{K}_i[\mathbf{y}^j] + \big(\mathbf{y}^i + \mathbf{y} + \mathbf{e}\big)\cdot\Delta_i\Big) + \sum_{j\neq i} \mathsf{M}_i[\mathbf{y}^j] + E_i \\
&= \sum_{j\neq i} \big(\mathsf{K}_i[\mathbf{y}^j] + \mathsf{M}_i[\mathbf{y}^j]\big) + \big(\mathbf{y}^i + \mathbf{y} + \mathbf{e}\big)\cdot\Delta_i + E_i \\
&= \sum_{j\neq i} \mathbf{y}^j\cdot\Delta_{i,j} + \big(\mathbf{y}^i + \mathbf{y} + \mathbf{e}\big)\cdot\Delta_i + E_i \\
&= \sum_{j\neq i} \mathbf{y}^j\cdot R_{i,j} + \big(\mathbf{y}^i + \mathbf{y} + \mathbf{e} + \sum_{j\neq i}\mathbf{y}^j\big)\cdot\Delta_i + E_i \\
&= \sum_{j\neq i} \mathbf{y}^j\cdot R_{i,j} + \mathbf{e}\cdot\Delta_i + E_i
\end{aligned}$$

If a malicious party P_i provides inconsistent global keys, then there exists $j_0, j_1 \in \mathcal{H}$ such that $R_{i,j_0} \neq R_{i,j_1}$. 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 view of the adversary, $\mathbf{y}^{j_0}$ 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 3. *The protocol Π_{aShare} shown in Figure 6 securely realizes the 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 $i \in \mathcal{H}$. Specifically, for any PPT adversary $\mathcal{A}$, we construct a PPT simulator $\mathcal{S}$ as follows:

1. In the initialization phase, $\mathcal{S}$ receives $\Delta_{i,j}$ for each $i \in \mathcal{M}$ and $j \neq i$ from $\mathcal{A}$. On behalf of every corrupt party $P_i \in \mathcal{M}$, $\mathcal{S}$ defines and sends $\Delta_i := \Delta_{i,j}$ for some $j \in \mathcal{H}$ 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 $\mathcal{A}$ and sent to $\mathcal{A}$. For each $i \in \mathcal{H}$, $\mathcal{S}$ 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 $j \in \mathcal{M}$ 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 returns the decision results from $\mathcal{F}_{\mathsf{aShare}}$ to $\mathcal{A}$. If $\mathcal{F}_{\mathsf{aShare}}$ aborts, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and halts. Otherwise, $\mathcal{S}$ continues to the simulation.

4. For each $i \in \mathcal{H}$, $\mathcal{S}$ 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 $i \in \mathcal{H}$ 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 $\mathcal{S}$ sends abort to $\mathcal{F}_{\mathsf{aShare}}$, and outputs whatever $\mathcal{A}$ outputs and halts.

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

Honest Parties: Upon receiving (LaAND) from all parties, generate random $\langle x\rangle, \langle y\rangle, \langle z\rangle$ such that $(\bigoplus_{i\in[n]} x^i)\wedge(\bigoplus_{i\in[n]} y^i) = \bigoplus_{i\in[n]} z^i$, and send them to the parties.

Corrupt Parties:

- Corrupt parties can choose their own randomness specifying the outputs received from the functionality.
- The adversary can send $(Q, q, \{R_i\}_{i\in\mathcal{H}}) \in \{0,1\}^\kappa \times \{0,1\} \times \{0,1\}^{|\mathcal{H}|\kappa}$ to $\mathcal{F}_{\mathsf{LaAND}}$, which checks

$$Q \oplus (\bigoplus_{i\in\mathcal{H}} x^i R_i) = (q \oplus \bigoplus_{i\in\mathcal{H}} x^i \mathsf{lsb}(R_i))(\bigoplus_{i\in\mathcal{H}} \Delta_i).$$

  If the check fails, the functionality sends `fail` to all parties and aborts. Otherwise, the functionality proceeds as if nothing has happened, and the adversary will know its guess is correct.

Leakage of global keys: The adversary may input $(\mathsf{leak}, i, S, \{\Delta'[k]\}_{k\in S})$. If P_i is honest, the functionality does the following:

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

Figure 11: Functionality for leaky authenticated AND triples with weak global keys

6. If there are two honest parties $j_0, j_1 \in \mathcal{H}$ such that $\Delta_{i,j_0} \neq \Delta_{i,j_1}$ for some $i \in \mathcal{M}$, then $\mathcal{S}$ aborts.
7. On behalf of every corrupt party P_i, $\mathcal{S}$ sends the corresponding shares, MACs and local keys received from $\mathcal{A}$ in the functionality $\mathcal{F}_{\mathsf{aBit}}$, to $\mathcal{F}_{\mathsf{aShare}}$.

By Lemma 5, we guarantee the probability that $\mathcal{S}$ aborts in Step 5 is negligible in κ. 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. $\mathcal{S}$ samples a dummy global key for every honest party to just perform the consistency check, and never uses these keys in any other place. In all, the real-world execution is statistically indistinguishable from the ideal-world execution, which completes the proof. □

C Efficient Protocol for Authenticated AND Triples

C.1 Improved Protocol for Leaky AND Triples

We first describe the functionality $\mathcal{F}_{\mathsf{LaAND}}$ for leaky authenticated AND triples in Figure 11. Similar to prior works, the adversary is allowed to guess a share $x^{i^*} \in \{0,1\}$ from an honest party P_{i^*}. An incorrect guess will be caught immediately, while a correct guess keep undetected. In more detail, an adversary $\mathcal{A}$ does not directly learn the share x^{i^*}, but instead is allowed to make a query on some linear combination of the values 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^*}.

We present a protocol Π_{LaAND} shown in Figure 12 that securely computes $\mathcal{F}_{\mathsf{LaAND}}$. In the protocol, we assume that $\mathcal{F}_{\mathsf{aShare}}$ generates global keys $\{\Delta_i\}_{i\in[n]}$ such that $\bigoplus_{i=1}^n \mathsf{lsb}(\Delta_i) = 1$, e.g., $\mathsf{lsb}(\Delta_i) = 1$ if $i \neq 1$ and $\mathsf{lsb}(\Delta_i) = n \mod 2$ if $i = 1$. Note that we add a tweak $i\|j\|t$ to the computation of H in the t-th execution of protocol Π_{LaAND}. It aims to prevent the attack in [GKWY20] that a malicious party P_j may send the same share and MAC in multiple protocol executions. Besides, we do not let the parties

Protocol Π_{LaAND}

In the t-th protocol execution, the parties generate a leaky AND triple as follows:

1. All parties send $(\mathsf{aShare}, 3)$ to $\mathcal{F}_{\mathsf{aShare}}$, which returns three random authenticated shares $\langle x \rangle, \langle y \rangle, \langle r \rangle$ to the parties, where $\langle x \rangle = \big([x^1]_1, \ldots, [x^n]_n\big)$, $\langle y \rangle = \big([y^1]_1, \ldots, [y^n]_n\big)$ and $\langle r \rangle = \big([r^1]_1, \ldots, [r^n]_n\big)$.
2. For each $i \in [n]$, P_i locally computes $\Phi_i := y^i \Delta_i \oplus \Big(\bigoplus_{k \neq i} (\mathsf{K}_i[y^k] \oplus \mathsf{M}_k[y^i])\Big)$.
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
   $$S_i := x^i \Phi_i \oplus \Big(\bigoplus_{k \neq i} (\mathsf{K}_i[x^k]_{\Phi_i} \oplus \mathsf{M}_k[x^i]_{\Phi_k})\Big) \oplus r^i \Delta_i \oplus \Big(\bigoplus_{k \neq i} (\mathsf{K}_i[r^k] \oplus \mathsf{M}_k[r^i])\Big).$$
   (b) Commit to $d_i := \mathsf{lsb}(S_i)$ by the Commit command of $\mathcal{F}_{\mathsf{Com}}$.
   (c) After receiving all commitments, open its commitment via the Open command of $\mathcal{F}_{\mathsf{Com}}$, and then compute $d := \bigoplus_{i=1}^n d_i$.
5. For each $i \in [n]$, P_i computes and commits to $T_i := S_i \oplus d\Delta_i$ by 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 the Open command of $\mathcal{F}_{\mathsf{Com}}$, and then check $\bigoplus_{i \in [n]} T_i = \mathbf{0}$. 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 $[z^1]_1 := [r^1]_1 \oplus d$.

Figure 12: Protocol for leaky authenticated AND triples with weak global keys

straightforwardly broadcast a bit d_i, and instead make them commit to d_i. This because in the security proof, the simulator needs to know the bits from the adversary, before sending a bit d_i on behalf of $P_i \in \mathcal{H}$.

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 generated, we can run ℓ times the Π_{LaAND} protocol in parallel, 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 particular, let $d_{i,t}$ be the bit committed by P_i in Step 4(b), and $T_{i,t}$ be the value committed by P_i in Step 5, in the t-th protocol execution. Then every party P_i can commit to $d_{i,1}, \ldots, d_{i,\ell}$ by making a single Commit request to $\mathcal{F}_{\mathsf{Com}}$, and commit to $T_{i,1}, \ldots, T_{i,\ell}$ via a single Commit request of $\mathcal{F}_{\mathsf{Com}}$. When using a random oracle H to implement $\mathcal{F}_{\mathsf{Com}}$ as shown in Section A.2, this means that P_i only needs to compute and broadcast a single value $H(i, \{d_{i,t}\}_{t \in [\ell]}, r_i)$ in Step 4(b) and $H(i, \{T_{i,t}\}_{t \in [\ell]}, r_i')$ in Step 5, where $r_i, r_i' \in \{0,1\}^\kappa$ are two randomness sampled by P_i.

C.1.1 Proof of Security for 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 overwhelming probability. See below.

Lemma 6. *Protocol $\mathcal{F}_{\mathsf{LaAND}}$ in Figure 12 is correct, when all parties are honest.*

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

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

Note that

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

Taking the above two equations, we have the following equation holds:

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

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

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

From $z^i=r^i$ and $z^1=r^1\oplus d$, we have $\left(\bigoplus_{i\in[n]}x^i\right)\wedge\left(\bigoplus_{i\in[n]}y^i\right)=d\oplus\left(\bigoplus_{i\in[n]}r^i\right)=\bigoplus_{i\in[n]}z^i$. □

Lemma 7. *If the honest parties do not abort, then $(\bigoplus_{i\in[n]}x^i)\wedge(\bigoplus_{i\in[n]}y^i)=\bigoplus_{i\in[n]}z^i$, where $\{z^i := r^i\}_{i\neq 1}$ and $z^1 := r^1\oplus d'$, the bit d' is computed in Step 4(c) of protocol Π_{LaAND}, and $\{x^i,y^i,r^i\}_{i=1}^n$ are defined from $\langle x\rangle,\langle y\rangle,\langle r\rangle$ which are 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 $i\in\mathcal{M}$, we define $R_{i,j} := U'_{i,j}\oplus U_{i,j}$ for each $j\in\mathcal{H}$ and $Q_i := T'_i\oplus T_i$. For each $j\in\mathcal{H}$, we also define $R_j := \bigoplus_{k\in\mathcal{M}}R_{k,j}$. Firstly, we show that if $d'=d=(\bigoplus_{i\in[n]}x^i)\wedge(\bigoplus_{i\in[n]}y^i)\oplus(\bigoplus_{i\in[n]}r^i)$, then $\bigoplus_{i\in[n]}z^i=(\bigoplus_{i\in[n]}x^i)\wedge(\bigoplus_{i\in[n]}y^i)$ holds with probability 1. Since $z^i=r^i$ for $i\neq 1$ and $z^1=r^1\oplus d'$, we have:

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

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 $i \in \mathcal{M}$, an honest party $P_j \in \mathcal{H}$ would compute $\mathsf{M}'_i[x^j]_{\Phi_i} := x^j \cdot U'_{i,j} \oplus \mathsf{H}(\mathsf{M}_i[x^j]) = x^j \cdot U_{i,j} \oplus \mathsf{H}(\mathsf{M}_i[x^j]) \oplus x^j \cdot R_{i,j} = \mathsf{M}_i[x^j]_{\Phi_i} \oplus x^j \cdot R_{i,j}$. Then P_j will compute $S'_j = S_j \oplus \left(\bigoplus_{k \in \mathcal{M}} x^j R_{k,j}\right) = S_j \oplus x^j \cdot R_j$. Note that we have $\bigoplus_{i \in [n]} T_i = \mathbf{0}$. Thus, we know that

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

To make $\bigoplus_{i\in[n]} T'_i$ be equal to 0, the adversary needs to find errors such that

$$\textstyle\bigoplus_{i\in\mathcal{M}} Q_i \oplus \bigoplus_{i\in\mathcal{H}} x^i R_i = \bigoplus_{i\in\mathcal{H}} \Delta_i \tag{1}$$

We here consider the case that $|\mathcal{H}| = 1$, because if $|\mathcal{H}| \geq 2$, the adversary $\mathcal{A}$ will have lower probability to guarantee the above equation (1) holds. We assume that $P_{i^*} \in \mathcal{H}$ is the unique honest party. If $\mathcal{A}$ succeeds to guess c bits of Δ_{i^*} via the leak command of $\mathcal{F}_{\mathsf{aShare}}$, then 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 $q/2^{\kappa-1-c} \cdot 1/2^c = q/2^{\kappa-1}$. □

Below, we prove the security of Π_{LaAND} in the following theorem.

Theorem 4. *Assume that* H *is a random oracle, then protocol* Π_{LaAND} *from Figure 12 securely realizes* $\mathcal{F}_{\mathsf{LaAND}}$ *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 $\mathcal{M}$. We construct a PPT simulator $\mathcal{S}$, which runs $\mathcal{A}$ as a subroutine, simulates $\mathcal{A}$'s view, and has access to the functionality $\mathcal{F}_{\mathsf{LaAND}}$.

Description of the simulation.

1. $\mathcal{S}$ emulates the functionality $\mathcal{F}_{\mathsf{aShare}}$, and for each $i \in \mathcal{M}$, $\mathcal{S}$ receives global key Δ_i and P_i's authenticated shares of $\langle x\rangle, \langle y\rangle, \langle r\rangle$ from $\mathcal{A}$. $\mathcal{S}$ samples a random bit $d \leftarrow \{0,1\}$, and then defines $[z^i]_i := [r^i]_i$ for each $i \neq 1$ and $[z^1]_1 := [r^1]_1 \oplus d$. For each $i \in \mathcal{M}$, $\mathcal{S}$ 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 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, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and aborts.

3. For each $i \in \mathcal{H}$, $\mathcal{S}$ 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 $i \in \mathcal{M}$, 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 $j \in \mathcal{H}, j \neq i$, $d_i := \mathsf{lsb}(S_i)$ 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 a semi-honest P_i with its authenticated shares and the messages $\{U_{j,i}\}_{j\neq i}$.

4. For each $i \in \mathcal{H}$, $\mathcal{S}$ acts as honest party P_i and sends $U_{i,j}$ sampled in previous step to P_j for each $j \neq i$. For each $i \in \mathcal{M}$, for every $j \in \mathcal{H}, j \neq i$, $\mathcal{S}$ 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 $i \in \mathcal{H}$, $\mathcal{S}$ computes $R_i := \bigoplus_{k\in\mathcal{M}} R_{k,i}$.

5. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{Com}}$ and receives d'_i for each $i \in \mathcal{M}$ from $\mathcal{A}$. Then, $\mathcal{S}$ computes $q_i := d'_i \oplus d_i$ and $q := \bigoplus_{i\in\mathcal{M}} q_i$. 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 overwhelming probability. Therefore, $\mathcal{S}$ sets $d' := d$. For each $i \in \mathcal{H}$, $\mathcal{S}$ samples $d'_i \leftarrow \{0,1\}$ such that $\bigoplus_{i\in[n]} d'_i = d'$. Then, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{Com}}$ and opens d'_i for each $i \in \mathcal{H}$ to all parties.

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

For each $i \in \mathcal{H}$, we assume that $\mathcal{A}$ guesses c_i bits of Δ_i 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 $i \in \mathcal{H}, j \neq i$, $U_{i,j}$ 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 kept secret. Therefore, $\{d'_i\}_{i\in\mathcal{H}}$ 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 $i \in \mathcal{H}$, $d'_i = d_i \oplus x^i \cdot \mathsf{lsb}(R_i)$. Due to $d'_i = d_i \oplus q_i$ for each $i \in \mathcal{M}$, we know that

$$\begin{aligned} d' &= \textstyle\bigoplus_{i\in[n]} d'_i = \bigoplus_{i\in\mathcal{H}} d'_i \oplus \bigoplus_{i\in\mathcal{M}} d'_i \\ &= \textstyle\bigoplus_{i\in[n]} d_i \oplus \bigoplus_{i\in\mathcal{H}} x^i \cdot \mathsf{lsb}(R_i) \oplus \bigoplus_{i\in\mathcal{M}} q_i \\ &= d \oplus \textstyle\bigoplus_{i\in\mathcal{H}} x^i \cdot \mathsf{lsb}(R_i) \oplus q. \end{aligned}$$

Based on the above equation, we have the following:

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

Therefore, $\bigoplus_{i\in[n]} T'_i = \mathbf{0}$ if and only if the following holds:

$$Q \oplus \left(\textstyle\bigoplus_{i\in\mathcal{H}} x^i R_i\right) = \left(q \oplus \textstyle\bigoplus_{i\in\mathcal{H}} x^i \mathsf{lsb}(R_i)\right)\left(\textstyle\bigoplus_{i\in\mathcal{H}} \Delta_i\right),$$

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 $i \in \mathcal{H}$, T'_i is chosen at random except for the least significant bit. We need to show that if the protocol does not abort, then $\{T'_i\}_{i\in\mathcal{H}}$ 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 $Q = \bigoplus_{i\in\mathcal{H}} x^i R_i$ and $q = \bigoplus_{i\in\mathcal{H}} x^i \mathsf{lsb}(R_i)$, 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 $d' = d \oplus q \oplus \bigoplus_{i\in\mathcal{H}} x^i \mathsf{lsb}(R_i) = d$. From $\mathcal{S}$'s simulation, we have:

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

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 $\bigoplus_{i\in[n]} T_i' = \mathbf{0}$ and $\mathsf{lsb}(T_i') = d_i' \oplus d' \cdot \mathsf{lsb}(\Delta_i)$ in the real protocol execution. When $|\mathcal{H}| = 1$ (i.e., only one party is honest), it is obvious that T_i' with $i \in \mathcal{H}$ is defined by the equation $\bigoplus_{i\in[n]} T_i' = \mathbf{0}$. In the following, we focus on the case that $|\mathcal{H}| \geq 2$. In particular, for each $i \in \mathcal{H}$, 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 $\mathsf{S} \subset \mathcal{H}$, $\bigoplus_{i\in\mathsf{S}} F_i$ is perfectly indistinguishable from a random value in $\{0,1\}^\kappa$. We use e to denote an honest party such that $e \in \mathcal{H}$ and $e \notin \mathsf{S}$. Such e always exists, as S is a proper subset of $\mathcal{H}$. We have the following equation 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, it is easy to see 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 picked uniformly at random by $\mathcal{F}_{\mathsf{aShare}}$, and is kept unknown for $\mathcal{A}$ as both $i, e \in \mathcal{H}$, $\bigoplus_{i\in\mathsf{S}} F_i$ is random and unknown for $\mathcal{A}$. Therefore, for any proper subset $\mathsf{S} \subset \mathcal{H}$, $\bigoplus_{i\in\mathsf{S}} S_i'$ is 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 4 for $i \in \mathsf{S}$. We have that for any proper subset $\mathsf{S} \subset \mathcal{H}$, $\bigoplus_{i\in\mathsf{S}} T_i'$ is indistinguishable from a random value except that $\mathsf{lsb}\left(\bigoplus_{i\in\mathsf{S}} T_i'\right)$ is fixed.

According to 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 overwhelming probability. Therefore, if honest parties do not abort, then protocol Π_{LaAND} will output a correct authenticated AND triple with overwhelming probability, while functionality $\mathcal{F}_{\mathsf{LaAND}}$ always output a correct AND triple. In conclusion, we complete the proof. □

C.2 From Leaky authenticated AND Triples to Authenticated AND Triples

Similar to all 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 se-

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

Honest Parties: Upon receiving (aAND, ℓ) from all parties, generate random $\{(\langle x_k \rangle, \langle y_k \rangle, \langle z_k \rangle)\}_{k \in [\ell]}$ such that $\left(\bigoplus_{i \in [n]} x_k^i\right) \wedge \left(\bigoplus_{i \in [n]} y_k^i\right) = \bigoplus_{i \in [n]} z_k^i$ for each $k \in [\ell]$, and send them to all parties.

Corrupt Parties: Corrupt parties can choose their own randomness received from the functionality.

Leakage of global keys: The adversary may input $(\mathsf{leak}, i, S, \{\Delta'[k]\}_{k \in S})$. If P_i is honest, the functionality does the following:

– If there exists some $k \in S$ such that $\Delta'[k] \neq \Delta_i[k]$, the functionality outputs `fail` to all parties and aborts.

– Otherwise, it outputs `success` to the adversary and proceeds as if nothing has happened.

Figure 13: Functionality $\mathcal{F}_{\mathsf{aAND}}$ for authenticated AND triples with weak global keys

curely computes a functionality $\mathcal{F}_{\mathsf{aAND}}$ shown in Figure 13. The details of Π_{aAND} are described in Figure 14. 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 with a certain probability, this has no impact on the security, by following the proof in Lemma 4.

Protocol Π_{aAND}

1. 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 $\{(\langle x_k \rangle, \langle y_k \rangle, \langle z_k \rangle)\}_{k \in [\ell']}$.
2. 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.
3. 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 AND triples in the same bucket, the parties just iterate by taking the result and combine it with the next element in the bucket.
4. All parties output the ℓ non-leaky AND triples.

Figure 14: The protocol Π_{aAND} for authenticated AND triples with weak global keys

D Security Proof of Protocol Π_{mpc}

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

D.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 in Π_{mpc}), when some parties are corrupted. Recall that Step 7 adopts a standard technique (i.e., authenticated Beaver triples [Bea92, BDOZ11]), and uses a hash function H to perform the amortized opening of authenticated shares in which the security is proved in Section A.4.

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 $\{\mathsf{L}^i_{w,\Lambda_w}\}_{i\neq 1}$.*

Proof. We prove this lemma by induction on the gates in the circuit.

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

Induction step. We can see that the lemma trivially holds for XOR gates. Thus, we focus on every 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:

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

Observe that for each $i \neq 1$

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

It is easy to verify that the following holds:

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

and

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

Each garbler P_i locally computes the 0-label $\mathsf{L}_{\gamma,0}$ as: $\mathsf{L}^i_{\gamma,0} := \mathsf{H}(\mathsf{L}^i_{\alpha,0},\gamma,0)\oplus\mathsf{H}(\mathsf{L}^i_{\beta,0},\gamma,1)\oplus(\bigoplus_{j\neq i}\mathsf{K}_i[\lambda^j_{\alpha\beta}])\oplus \lambda^i_{\alpha\beta}\Delta_i \oplus (\bigoplus_{j\neq i}\mathsf{K}_i[\lambda^j_\gamma])\oplus\lambda^i_\gamma\Delta_i$. Thus, we conclude:

$$\begin{aligned}
\mathsf{L}^i_{\gamma,0}\oplus\mathsf{L}^i_{\gamma,\Lambda_\gamma} = \;& \Lambda_\alpha\Lambda_\beta\Delta_i\oplus\Lambda_\alpha\lambda_\beta\Delta_i\oplus\Lambda_\beta\lambda_\alpha\Delta_i\oplus(\textstyle\bigoplus_{j\neq i}\mathsf{K}_i[\lambda^j_{\alpha\beta}]\oplus\bigoplus_{j\neq i}\mathsf{M}_i[\lambda^j_{\alpha\beta}]\oplus\lambda^i_{\alpha\beta}\Delta_i)\\
&\oplus(\textstyle\bigoplus_{j\neq i}\mathsf{K}_i[\lambda^j_\gamma]\oplus\bigoplus_{j\neq i}\mathsf{M}_i[\lambda^j_\gamma]\oplus\lambda^i_\gamma\Delta_i)\\
=\;& \Lambda_\alpha\Lambda_\beta\Delta_i\oplus\Lambda_\alpha\lambda_\beta\Delta_i\oplus\Lambda_\beta\lambda_\alpha\Delta_i\oplus\lambda_{\alpha\beta}\Delta_i\oplus\lambda_\gamma\Delta_i\\
=\;& \Lambda_\alpha\Lambda_\beta\Delta_i\oplus\Lambda_\alpha\lambda_\beta\Delta_i\oplus\Lambda_\beta\lambda_\alpha\Delta_i\oplus\lambda_\alpha\lambda_\beta\Delta_i\oplus\lambda_\gamma\Delta_i\\
=\;& \big((\Lambda_\alpha\oplus\lambda_\alpha)\wedge(\Lambda_\beta\oplus\lambda_\beta)\oplus\lambda_\gamma\big)\Delta_i=\Lambda_\gamma\Delta_i,
\end{aligned}$$

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

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

□

Lemma 9. *For every PPT adversary $\mathcal{A}$ which corrupts a set of parties such that $P_1\in\mathcal{M}$ is corrupted, with probability at most $q/2^{\kappa-1}$, the protocol execution does not abort and $\mathcal{A}$ learns both garbled labels for some wire generated by some honest party P_i, where q is an upper bound of the number of queries to* H.

Proof. Clearly, $\mathcal{A}$ learns both garbled labels from honest party P_i 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 via the leak command of $\mathcal{F}_{\mathsf{prep}}$, then the protocol execution will abort except with probability $1/2^{c_i}$.

It is clear that the opening process of authenticated bits/shares does not reveal any information of Δ_i, since all MACs received by $\mathcal{A}$ from $\mathcal{F}_{\mathsf{prep}}$ do not include any information on Δ_i. 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 H. Therefore, 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 $q/2^{\kappa-1-c_i}$, where Δ_i has entropy at least $\kappa-1-c_i$ if $i=2$ and $\kappa-c_i$ otherwise.

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 9(a) and λ_w is defined by $\mathcal{F}_{\mathsf{prep}}$. If any PPT adversary $\mathcal{A}$ corrupts a set of parties such that $P_1\in\mathcal{H}$ is honest, then either P_1 aborts, or P_1 outputs $y^1=f_1(x^1,\ldots,x^n)$ with probability at least $1-(\epsilon+\frac{q+2}{2^\kappa})$, provided that $\mathbf{H}$ is ϵ-almost universal and $\mathcal{A}$ makes at most q queries to H, where f_1 denotes the P_1's output of 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 $1-1/2^\kappa-\epsilon$, where $\epsilon=|\mathcal{C}|/2^\kappa$ if a polynomial hash is used to instantiate $\mathbf{H}$.

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_γ^i for $i \in [n]$, we have

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

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

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

For each $i \neq 1$, we use $\mathbf{c}_i$ to denote the correct value computed with $\mathbf{H}$ and the MACs held by P_i, and $\hat{\mathbf{c}}_i$ to denote the value sent by malicious P_i. Thus, $\sum_{i\neq 1} \hat{\mathbf{c}}_i = \sum_{i\neq 1} \mathbf{c}_i + \mathbf{e}$, where $\mathbf{e}$ is some additive error of $\mathcal{A}$'s choice. Note that $\mathbf{c}_1$ is correct, as P_1 is honest. Since $\mathbf{H}$ is additively homomorphic, we have

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

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

Below, we analyze the probability that $\mathbf{H}(\{t_w\}_{w\in\mathcal{W}}) \neq \mathbf{0}$. We assume that the adversary $\mathcal{A}$ leaks c_1 bits of Δ_1 by the leak command of $\mathcal{F}_{\mathsf{prep}}$. In this case, the protocol will abort with probability 2^{-c_1}. Then the remaining $\kappa - c_1$ bits of Δ_1 are uniformly random in $\mathcal{A}$'s view. Thus, $\mathbf{e}$ 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 $\Delta_1 = \left(\mathbf{H}(\{t_w\}_{w\in\mathcal{W}})\right)^{-1} \cdot \mathbf{e}$ 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 $\mathbf{H}(\{t_w\}_{w\in\mathcal{W}}) \neq \mathbf{0}$.

As a result, if P_1 does not abort, we have $\mathbf{H}(\{t_w\}_{w\in\mathcal{W}}) = \mathbf{0}$ with probability $1 - 2^{-\kappa}$. Since $\{t_w\}_{w\in\mathcal{W}}$ are independent of $\mathbf{H}$ and $\mathbf{H}$ is ϵ-almost universal, the probability that there exists a $w \in \mathcal{W}$ such that $t_w \neq 0$ is at most ϵ. Overall, with probability at least $1 - 1/2^\kappa - \epsilon$, $t_w = 0$ 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, the probability that there exists a corrupt party P_j flips its share $\lambda^j_{w'}$ for some $w' \in \mathcal{O}_1$ is $(q+1)/2^\kappa$, according to Lemma 4. 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 $1/2^\kappa + \epsilon$. In conclusion, if P_1 does not abort, then $y^1_w = \Lambda_w \oplus \lambda_w$ is correct for each $w \in \mathcal{O}_1$, except with probability $(q+2)/2^\kappa + \epsilon$. □

Lemma 11. *For every PPT adversary $\mathcal{A}$ corrupting a subset of parties, every honest party P_i either aborts, or outputs $y^i = f_i(x^1, \ldots, x^n)$ with probability at least $1 - 6q/2^\kappa$, where f_i denotes the output of f to P_i and q is an upper bound of the number of queries to* H.

Proof. We first prove that $P_i \in \mathcal{H}$ 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 does not abort, then $h'_i = \mathsf{H}(\{\mathsf{L}^i_{w,0} \oplus \Lambda'_w \Delta_i\}_{w \in \mathcal{W}})$. We can construct an algorithm that controls the random oracle H and extracts $\{\mathsf{L}^i_{w,\Lambda'_w}\}_{w \in \mathcal{W}}$ from h'_i and the query/response pairs of H. Since H is a random oracle, the probability that $\mathcal{A}$ finds a target collision is $q/2^\kappa$. Therefore, with probability $1 - q/2^\kappa$, $\mathsf{L}^i_{w,\Lambda'_w} = \mathsf{L}^i_{w,0} \oplus \Lambda'_w \Delta_i$ for each $w \in \mathcal{W}$ is learned by $\mathcal{A}$. In addition, $\mathcal{A}$ has learned $\mathsf{L}^i_{w,\Lambda_w} = \mathsf{L}^i_{w,0} \oplus \Delta_w \Delta_i$ 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 $\mathsf{L}^i_{w,0}$ and $\mathsf{L}^i_{w,1}$ 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.

For each $w \in \mathcal{O}_i$, let $\hat{\lambda}^j_w$ be the share sent by a malicious party P_j in Step 13(a), and λ^j_w be the correct share from $\mathcal{F}_{\mathsf{prep}}$. In the Open process of Step 13(a), we analyze the probability that there exists some $j \in \mathcal{M}$ and $w \in \mathcal{O}_i$ such that $\hat{\lambda}^j_w \neq \lambda^j_w$. If P_i does not abort, then the value τ'_i sent by $\mathcal{A}$ on behalf of P_j is equal to $\mathsf{H}(\text{“open”}, \{\mathsf{K}_i[\lambda^j_w] \oplus \hat{\lambda}^j_w \Delta_i\}_{w \in \mathcal{O}_i})$. If $\mathcal{A}$ makes a query $\mathbf{z}$ to H such that $\tau'_i = \mathsf{H}(\mathbf{z})$ and $\mathbf{z} \neq (\text{“open”}, \{\mathsf{K}_i[\lambda^j_w] \oplus \hat{\lambda}^j_w \Delta_i\}_{w \in \mathcal{O}_i})$, $\mathcal{A}$ finds a target collision for random oracle H, which occurs with probability $q/2^\kappa$. Now, we assume that $\mathcal{A}$ does not find a target collision. We construct an algorithm that controls H and extracts $\{\mathsf{M}_i[\hat{\lambda}^j_w]\}_{w \in \mathcal{O}_i}$ from τ'_i and the records of H queries. By the assumption, we have that $\mathsf{M}_i[\hat{\lambda}^j_w] = \mathsf{K}_i[\lambda^j_w] \oplus \hat{\lambda}^j_w \Delta_i$. If $\hat{\lambda}^j_w \neq \lambda^j_w$ for some $w \in \mathcal{O}_i$, then $\mathcal{A}$ learns $\Delta_i := \mathsf{M}_i[\hat{\lambda}^j_w] \oplus \mathsf{M}_i[\lambda^j_w]$, as $\mathcal{A}$ also holds the MAC $\mathsf{M}_i[\lambda^j_w] = \mathsf{K}_i[\lambda^j_w] \oplus \lambda^j_w \Delta_i$. However, from the proof of Lemma 9, the probability that $\mathcal{A}$ learns Δ_i is at most $q/2^{\kappa-1}$ in the random oracle model. Therefore, P_i will obtain a correct wire mask λ_w for each $w \in \mathcal{O}_i$, except with probability $3q/2^\kappa$.

In conclusion, if P_i does not abort, then $y^i_w = \Lambda_w \oplus \lambda_w$ is correct for each $w \in \mathcal{O}_i$, except with probability $6q/2^\kappa$. □

D.2 Main Proof of Security

Theorem 2. *Let $f : \{0,1\}^{n|\mathcal{I}|} \to \{0,1\}^{n|\mathcal{O}|}$ be an n-party functionality. Then the protocol Π_{mpc} shown in Figures 7 and 8 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.*

Given the above Lemmas 8–11, the proof of Theorem 2 is relatively easy. Below, we present the details of the proof.

Proof. Let $\mathcal{A}$ be a PPT adversary who corrupts a subset of parties $\mathcal{M}$. Recall that $\mathcal{H}$ is a set of honest parties such that $\mathcal{H} = [n] \backslash \mathcal{M}$. 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 $\mathcal{M}$, $\mathcal{S}$ corrupts the same parties in the ideal world, and then 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 the inputs of $\mathcal{A}$ and all the outputs. $\mathcal{S}$ acts as every honest party $P_i \in \mathcal{H}$ 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 $P_i \in \mathcal{H}$, $\mathcal{S}$ adopts $x^i := \mathbf{0}$ 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 $P_i \in \mathcal{M}$, 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 $P_i \in \mathcal{M}$, $\mathcal{S}$ 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 $\{\tilde{x}^i := \mathbf{0}\}_{i \in \mathcal{H}}$ and $\{\tilde{x}^i := x^i\}_{i \in \mathcal{M}}$. Then $\mathcal{S}$ chooses a $j^* \in \mathcal{H}$, and then for each $i \in \mathcal{M}, w \in \mathcal{O}_i$, 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 $i \in \mathcal{M}$, $\mathcal{S}$ acts as 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, we prove that the protocol execution in the $\mathcal{F}_{\mathsf{prep}}$-hybrid model is indistinguishable from the ideal world execution by a sequence of games.

$\mathbf{Hybrid_0}$. This is the same as the protocol execution shown in Figures 7 and 8, where the actual inputs $\{x^i\}_{i \in \mathcal{H}}$ are used for honest parties.

$\mathbf{Hybrid_1}$. This is the same as $\mathbf{Hybrid_0}$, except that $\mathcal{S}$ plays the role of honest parties $\{P_i\}_{i \in \mathcal{H}}$.

$\mathbf{Hybrid_1}$ is essentially the same as $\mathbf{Hybrid_0}$.

$\mathbf{Hybrid_2}$. This is the same as $\mathbf{Hybrid_1}$, except that a) for each $i \in \mathcal{M}, w \in \mathcal{I}_i$, $\mathcal{S}$ receives a public value Λ_w from $\mathcal{A}$ and computes $x_w^i := \Lambda_w \oplus \lambda_w$; b) for each $i \in \mathcal{M}$, $\mathcal{S}$ 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 $\mathbf{Hybrid_1}$ and $\mathbf{Hybrid_2}$ are identical. If P_1 is honest, then the outputs obtained by P_1 in two hybrids are the same except with negligible probability from Lemma 8 and Lemma 10. If P_i is honest for each $i \in \mathcal{H} \backslash \{1\}$, then the outputs obtained by P_i in two hybrid games are the same except with negligible probability by Lemma 11. Therefore, the distributions in $\mathbf{Hybrid_1}$ and $\mathbf{Hybrid_2}$ are indistinguishable.

$\mathbf{Hybrid_3}$. This is the same as $\mathbf{Hybrid_2}$, except that $\mathcal{S}$ executes as follows:

1. Use $\{x^i = \mathbf{0}\}_{i \in \mathcal{H}}$ as the inputs of honest parties in Step 9.
2. Compute $(\tilde{y}^1, \ldots, \tilde{y}^n) := f(\tilde{x}^1, \ldots, \tilde{x}^n)$, where $\tilde{x}^i := \mathbf{0}$ for each $i \in \mathcal{H}$ and $\tilde{x}^i := x^i$ for each $i \in \mathcal{M}$.
3. Choose a $j^* \in \mathcal{H}$; for each $i \in \mathcal{M}, w \in \mathcal{O}_i$, define $\tilde{\lambda}_w^{j^*} := \lambda_w^{j^*} \oplus y_w^i \oplus \tilde{y}_w^i$; and 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 $i \in \mathcal{M}$, act as honest party P_{j^*} and open $\{\tilde{\lambda}_w^{j^*}\}_{w \in \mathcal{O}_i}$ to P_i.

We first prove that for every honest party P_i, its share λ_w^i for each wire w in the circuit are kept secret in $\mathcal{A}$'s view, before these shares are revealed in the phases of input and output processing. Here we do not include 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 $i \geq 2$, 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 $\mathcal{G}_{\gamma,1}^i$ 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 negligible probability. Besides, each garbled row $G_{\gamma,uv}^{i,j}$ for each $u, v \in \{0, 1\}$ 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 two garbled rows, $\mathcal{A}$ needs to learn both garbled labels for some input wire. By Lemma 9, this happens with negligible probability. Therefore, $\mathcal{A}$ does not learn the shares λ_α^i and λ_β^i for the input wires. In the phase of checking public values, if P_i does not abort, the public values on the output wires of all AND gates are correct with overwhelming probability, according to the proof of Lemma 11. Therefore, the value $\mathbf{c}_i$ 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 $i \in \mathcal{H}, w \in \mathcal{I}_i$, we have proved that λ_w^i is uniformly random and unknown for $\mathcal{A}$. Therefore, the distributions of the public values $\{\Lambda_w\}_{w \in \bigcup_{i \in \mathcal{H}} \mathcal{I}_i}$ in $\mathbf{Hybrid_2}$ and $\mathbf{Hybrid_3}$ 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 $P_{j^*} \in \mathcal{H}$, and both $\lambda_w^{j^*}$ and $\tilde{\lambda}_w^{j^*}$ are uniformly random. Therefore, for each $i \in \mathcal{M}, w \in \mathcal{O}_i$, $\tilde{\lambda}_w^{j^*}$ sent by $\mathcal{S}$ in $\mathbf{Hybrid_3}$ has the same distribution as $\lambda_w^{j^*}$ sent by honest party P_{j^*} in $\mathbf{Hybrid_2}$.

For each $w \in \mathcal{W}$, the public value Λ_w is uniformly random and has the same distribution in $\mathbf{Hybrid_2}$ and $\mathbf{Hybrid_3}$, as $\{\lambda_w^i\}_{i \in \mathcal{H}}$ are uniformly random and not known to $\mathcal{A}$. If $P_1 \in \mathcal{M}$, P_1 is able to learn only one garbled label for each wire, according to Lemma 9. Thus, P_1 can open only one of four garbled rows $G_{\gamma,00}^{i,j}, G_{\gamma,01}^{i,j}, G_{\gamma,10}^{i,j}, G_{\gamma,11}^{i,j}$ for each $i \in \mathcal{H}$ 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, garbled labels obtained by P_1 are indistinguishable in two hybrids.

By Lemma 10, if $P_1 \in \mathcal{H}$ does not abort in Step 12, then $\Lambda_w = \tilde{y}_w^i \oplus (\bigoplus_{j \in [n]} \lambda_w^j)$ for each $w \in \mathcal{O}_i$ with overwhelming probability. Therefore, for each $i \in \mathcal{M}$ and $w \in \mathcal{O}_i$, $\Lambda_w \oplus (\bigoplus_{j \neq j^*} \lambda_w^j) \oplus \tilde{\lambda}_w^{j^*} = \Lambda_w \oplus (\bigoplus_{j \in [n]} \lambda_w^j) \oplus y_w^i \oplus \tilde{y}_w^i = y_w^i$, which means that $\mathcal{A}$ will obtain the correct output. If $P_1 \in \mathcal{M}$, $\mathcal{A}$ will also get the correct output for each $i \in \mathcal{M}$, due to the setting of the shares $\{\tilde{\lambda}_w^{j^*}\}_{w \in \mathcal{O}_i}$ of honest party P_{j^*}.

In conclusion, $\mathbf{Hybrid_3}$ is indistinguishable from $\mathbf{Hybrid_2}$.

$\square$

E More Efficiency Comparison

In this section, we further compare our protocol with the state-of-the-art actively secure constant-round two-party/multi-party computation protocols for computation security parameter $\kappa = 128$ and statistical security parameter $\rho = 64$.

From Table 3, we see that our protocol significantly improves both preprocessing phases, while it does not increase the online communication. Specifically, in the function-independent phase, our protocol gives an $\approx 1.4\times$ improvement for both single execution and 1024 executions. In the function-dependent phase,

#Parties	Protocol	Ind. (MB)		Dep. (MB)	Online (KB)	Total (MB)	
		#1	#1024			#1	#1024
$n = 2$	KRRW [KRRW18]	3.8	2.6	0.2	5.0	4.0	2.8
	Ours	2.7	1.9	0.2	4.2	2.9	2.1
$n = 3$	WRK [WRK17b]	7.6	5.2	1.1	6.3	8.7	6.3
	Ours	5.5	3.7	0.66	6.2	6.2	4.4
$n = 5$	WRK [WRK17b]	15.2	10.4	2.0	10.4	17.2	12.4
	Ours	11.0	7.5	1.5	10.3	12.5	9.0

Table 3: Comparison between our MPC protocol and the best known protocols in terms of communication overhead for secure AES evaluation.

our protocol provides $1.67\times$ improvement for $n = 3$ and $1.33\times$ improvement for $n = 5$. Overall, our optimizations result in $\approx 1.4\times$ improvement in terms of communication cost.

#Parties	Protocol	Hamming Distance			Sorting		
		Ind.	Dep.	Online	Ind.	Dep.	Online
$n = 2$	KRRW [KRRW18]	840.7	67.9	67.9	3859.1	327.8	5.5
	Ours	589.0	67.9	67.4	2791.7	327.8	4.2
$n = 3$	WRK [WRK17b]	1886.9	336.6	101.5	7744.9	1640.8	6.4
	Ours	1211.6	202.6	100.9	5583.4	987.8	6.3
$n = 5$	WRK [WRK17b]	4176.5	606.1	169.1	15515.0	2954.5	10.6
	Ours	2423.3	472.1	168.0	11166.8	2301.5	10.5

Table 4: Communication cost of our and prior best protocols for computing hamming distance and sorting. All numbers are in megabytes (MB) with a single execution.

In Table 4, we provide the comparison of communication cost for evaluating large circuits. In the function-independent phase of secure hamming distance evaluation, our protocol provides about $1.43\times$ improvement for two-party case, $1.56\times$ improvement for three-party setting, and $1.72\times$ improvement for five-party case. For sorting, our protocol achieves $\approx 1.4\times$ improvement in the function-independent phase. In the function-dependent phase, our protocol obtains $1.66\times$ improvement for three-party case, and $1.28\times$ improvement for five-party case. In particular, our protocol reduces the total communication by about 251 MB $\sim$ 1.8 GB for evaluating hamming distance and $1 \sim 5$ GB for secure sorting evaluation, when there are $2 \sim 5$ parties running the protocol.