Leakage-Tolerant Computation without Leakage-Free Setup

Yaohua Ma

Tsinghua University
ma-yh21@mails.tsinghua.edu.cn

Yifan Song

Tsinghua University and Shanghai Qi Zhi Institute
yfsong@mail.tsinghua.edu.cn

Abstract

Leakage-tolerant multi-party computation (MPC) ensures that leakage from each honest party's internal state reveals no more than the leakage from honest parties' individual inputs and outputs, beyond the standard security guarantee of MPC. Previous constructions either require a leakage-free preprocessing, or rely on indistinguishability obfuscation in the common reference string (CRS) model and only achieve semi-honest security.

In the computational setting, assuming only an augmented non-committing encryption scheme, we show that every adaptively well-formed functionality admits a semi-honest leakage-tolerant MPC protocol, without any leakage-free preprocessing execution or CRS. Under the decisional Diffie–Hellman assumption, we further extend this result to malicious security in the CRS model for all deterministic functionalities. The limitation of deterministic functions is subject to an impossibility result of leakage-based coin-tossing protocols.

In the information-theoretic setting, we show that secure erasure enables statistically secure leakage-tolerant MPC of arbitrary functionalities assuming strictly less than half of the participants are maliciously corrupted. We complement this construction with an impossibility result: without secure erasure, general-purpose statistically secure leakage-tolerant MPC with leakage-oblivious simulation, an important property to enable universal composition, is impossible in the plain model, even when all parties are honest.

As a further application, we also obtain information-theoretic leakage-tolerant circuits against bounded communication leakage, while all prior works are restricted only to the weaker variant of leakage-resilient circuits.

1 Introduction

The main theme of modern cryptography is to protect secrets under partial information leakage. In the context of storing secrets, this has led to a rich line of works for secret sharing [Sha79], which provides secrecy guarantee even if a bounded number of shares are revealed to an adversary. For the more general goal of protecting secrets over computation, this has led to the notion of secure multi-party computation (MPC) [Yao86, GMW87], which ensures the privacy of honest parties' inputs and outputs even if a bounded number of parties are corrupted by an adversary. The standard security guarantee of these notions usually treats the unrevealed shares or honest parties' internal states as inaccessible to an adversary. On the other hand, in practice, side-channel attacks may allow an adversary to learn partial information about inputs, random coins, and intermediate computations via physical interaction with the hardware or the machine that carries the secret data or computation of an honest party. Such a mismatch creates a security gap between theory and practice.

Leakage-Tolerant Computation and Leaky-UC Model. In the work of Garg, Jain and Sahai [GJS11], the authors initiated the study of zero-knowledge proofs under side-channel leakage. Then, the work of Bitansky, Canetti and Halevi [BCH12] introduced the leaky universal composable (leaky-UC) framework that captures side-channel leakage in theory. At a high level, the leaky-UC model extends the standard UC model by allowing an adversary to obtain partial leakage about an honest party's view, which is formalized by the output of applying a leakage function specified by the adversary on the entire view of an honest party. A protocol is said to be secure in the leaky-UC model, or *leakage-tolerant*, if the real world execution of the protocol reveals no more than comparable leakage from an ideal evaluation of the desired function, where an honest party's view in the ideal world only contains its function input and output. This requires a simulator in the ideal world that simulates the leakage in the real world using only each honest party's input and output *individually*. Note that, such a relaxation of the security guarantee in the ideal world is unavoidable given that private inputs and outputs themselves may leak.

In short, a leakage-tolerant MPC (or LTMPC) protocol can be viewed as an ideal and leak-free black-box, in the sense that any leakage from honest parties' views which can be obtained by the adversary during the protocol execution can also be simulated by leakage from each honest party's function input and output only.

To handle arbitrary leakage functions from the adversary and support universal composition, [BCH12] further considers *leakage-oblivious simulation*: the simulator supplies a state translation function for each honest party which, independently of the leakage function, generates the entire simulated view of the corresponding honest party given its function input and output. Then the leakage can be simulated by evaluating the leakage function directly on the simulated view, preserving the locality and number of leaked bits. In this work, we will focus only on LTMPC with leakage-oblivious simulation.

In the literature, [BCH12] gave feasibility results for basic computation tasks including secure communication and oblivious transfer in the common reference string (CRS) model. After that, follow-up works [BGJK12, BDSL14] obtained general-purpose leakage-tolerant MPC assuming a one-time leak-free preprocessing for all executions and secure erasure. Then, [CPV17a] removes the need of secure erasure but requires a leak-free preprocessing for each execution. The work [CPV17b] further removes the need of leak-free preprocessing and sets in the global CRS model (where the ideal simulator is not able to program the CRS), but relies on the expensive assumption of indistinguishability obfuscation (iO) and only achieves semi-honest security.

As we can see, known results in the literature for general-purpose LTMPC either require leak-free preprocessing or rely on the strong assumption of iO with only semi-honest security. This leads to our following question:

> *Can general leakage-tolerant MPC be realized in the plain model, with leakage-oblivious simulation and without leakage-free stage anywhere in the execution, without assuming iO?*

1.1 Our Contribution

In this work, we answer the above question affirmatively for both computational security and information-theoretic security. Concretely, in the computational setting, we obtain the following result for semi-honest security. Compared with [CPV17b], our result does not rely on the assumption of iO and further removes the need of CRS for semi-honest security.

Theorem 1.1 (Computationally Secure Semi-Honest LTMPC)**.** *Let ℓ be an a priori leakage bound. Assume that augmented non-committing encryption exists. Then for all $n \geq 2$, every n-party adap-*

tively well-formed functionality $\mathcal{F}$ admits a computationally secure LTMPC protocol that tolerates up to $n-1$ semi-honest corruptions in the plain model.

For malicious security, we give a feasibility result for deterministic functionalities in the CRS model as stated below. We note that for general computation with a dishonest majority, a setup assumption beyond authenticated communication is already necessary for malicious security in the standard UC model, even without leakage [CF01, CKL03]. Our use of a CRS addresses this inherent UC limitation, while the protocol still requires no leakage-free preprocessing. In addition, as [BCH12], the CRS in our construction can be reused across multiple executions. The other limitation of only supporting deterministic functionalities is also justified by the previous work [BGJ+13], which shows that maliciously secure coin-tossing encounters a leakage-based obstruction even when a common reference string is available.

Theorem 1.2 (Computationally Secure Malicious LTMPC)**.** *Let ℓ be an a priori leakage bound. Assume that the computational hardness of the decisional Diffie-Hellman problem (DDH). Then for all $n \geq 2$, every n-party deterministic functionality $\mathcal{F}$ admits a computationally secure LTMPC protocol with abort that tolerates up to $n-1$ malicious corruptions in the CRS model.*

In the information-theoretic setting, assuming secure erasure, we obtain the following result for malicious security.

Theorem 1.3 (Unconditionally Secure LTMPC)**.** *Let ℓ be an a priori leakage bound. Then for all $n \geq 2$, every n-party functionality $\mathcal{F}$ admits an unconditionally secure LTMPC protocol with abort that tolerates up to $\lfloor \frac{n-1}{2} \rfloor$ malicious corruptions in the secure erasure model.*

We justify the need of secure erasure by establishing the following negative result: without secure erasure, LTMPC with information-theoretic security and leakage-oblivious simulation is impossible even if there is no corrupted party. (Note that in this case, the adversary can still obtain leakage from each honest party's view. Obtaining a construction for the case of no corruption is not trivial.)

Application of Our Result in Leakage-Tolerant Circuits. A relevant notion of leakage-tolerant circuits (LTCs) introduced by Ishai and Song [IS24] studies the security guarantee of an unprotected circuit under leakage. For a leakage class $\mathcal{L}$, a leakage-tolerant circuit implementation C of a function f ensures that any leakage attack from $\mathcal{L}$ applied to circuit wires in C can be simulated by some leakage attack from the same leakage class $\mathcal{L}$ applied only on circuit inputs and outputs. In short, an LTC C can be viewed as a black-box implementation of f under leakage from $\mathcal{L}$ in the sense that attacking internal wires of C is equivalent to attacking the input and output wires of C directly.

Starting from an LTMPC protocol, when writing each party's computation as a circuit, we naturally obtain a leakage-tolerant circuit against the following leakage class $\mathcal{L}$:

- The circuit wires are partitioned into n disjoint parts, where each part corresponds to one party's computation in the LTMPC protocol.
- A leakage function $L \in \mathcal{L}$ acts as an adversary who (1) obtains the entire inputs from a subset of wire partitions (corresponding to corrupted parties' view), (2) adaptively queries leakage from each partition, and (3) outputs its view as the function output.

Such a leakage class can be viewed as an extension of the *bounded communication leakage* (BCL) introduced by [GIM+16], which corresponds to the case of 2 parties with no corruption in LTMPC. Previous works [GIM+16, IS25] only obtain the weaker *leakage-resilient* circuit against BCL, where

a leakage-resilient circuit further assumes a leak-free input encoder I and output decoder O such that for any input $\boldsymbol{x}$, $f(\boldsymbol{x}) = O(C(I(\boldsymbol{x})))$, and it requires that the leakage attack applied only to the wires of C reveals no information about the input $\boldsymbol{x}$. (In [IS24], it has been shown that a leakage-resilient circuit can be constructed from a leakage-tolerant circuit together with a leakage-resilient encoding scheme. The other direction, however, is not known.)

As a by-product, we obtain the first construction of LTC against BCL as stated below, which may be of independent interest.

Theorem 1.4 (Bounded Communication Leakage-Tolerant Circuits)**.** *Let ℓ be an a priori leakage bound. Then every function f that maps a 2-partitioned input $(\boldsymbol{x}_1, \boldsymbol{x}_2)$ to 2-partitioned output $(\boldsymbol{y}_1, \boldsymbol{y}_2)$ can be computed by an information-theoretic LTC against BCL.*

2 Technical Overview

This section explains the techniques used to construct leakage-tolerant multi-party computation (LTMPC) protocols without a leakage-free preprocessing phase.

2.1 Framework for LTMPCs.

The pioneering work by Goyal, Ishai, Maji, Sahai and Sherstov [GIM+16] studies a weaker model which we refer to as *leakage-resilient computation*. In this model, the function input is initially secret shared among all parties using a secret sharing scheme that is resilient to arbitrary bounded-size leakage from each individual share, and the output of the protocol is also only a secret sharing of the function output. The security only requires that when the input sharing is randomly generated without leakage, any bounded-size (and possibly adaptive) leakage from each party's view reveals no information about the function input and output. In [GIM+16], they refer to such a leakage as bounded communication leakage (BCL) [1]. In the following, we will use n-party *bounded communication leakage-resilient secret sharing* or n-BCLSS to denote a secret sharing scheme among n parties that is resilient to bounded communication leakage.

In [GIM+16], the authors give a generic framework for secure two-party leakage-resilient computation with no corruption that computes from secretly shared function input to secretly shared function output [2]. Their construction makes use of a *parity-resilient circuit* (i.e., a leakage-resilient circuit against parity leakage where each leaked bit is the parity of some subset of wires in the circuit) and works as follows:

- Let C_{pr} be a parity-resilient circuit implementation of $\mathcal{F}$ that makes use of a parity-resilient encoding scheme $(\mathsf{Enc}_{\mathsf{pr}}, \mathsf{Dec}_{\mathsf{pr}})$ as the input encoder I and output decoder O.
- Parties hold an additive sharing of $\mathsf{Enc}_{\mathsf{pr}}(\boldsymbol{x})$, which is viewed as a 2-BCLSS of $\boldsymbol{x}$.
- The parties aim to compute an additive sharing of $\mathsf{Enc}_{\mathsf{pr}}(\boldsymbol{y})$. To accomplish this, both parties jointly compute an additive secret sharing of wire values of C_{pr} (say, $(R, R \oplus W[C_{\mathsf{pr}}])$), in which

[1] In [GIM+16], the leakage is defined as the transcript of a communication-bounded interactive protocol among (malware of) parties where each party takes as input its entire view and the communication complexity of the protocol is bounded by some parameter ℓ. We note that this kind of leakage is captured by the leaky-UC model where an adversary can make adaptive leakage query to each party's view and thus is able to emulate such a communication-bounded interactive protocol. We will continue to use the name of bounded communication leakage but specifically refer to the type of leakage in the leaky-UC model. See Section 3.3 for details.

[2] More precisely, [GIM+16] constructs a leakage-resilient circuit against BCL that can be translated into the language of leakage-resilient computation.

every addition gate can be locally computed by simply accumulating the corresponding shares, and every multiplication gate is implemented by invocations of semi-honest leakage-tolerant oblivious transfer [BCH12, GIM+16].

Their analysis rests on the *small-biased masking lemma* which connects parity leakage with bounded communication leakage. Concretely, the small-biased masking lemma states that if two distributions X and Y cannot be distinguished by any single-bit parity leakage with an advantage more than ϵ, then the 2-party additive sharings $(R, X \oplus R)$ and $(R, Y \oplus R)$ cannot be distinguished by any ℓ-bit BCL function with an advantage more than $2^{\ell/2}\epsilon$. By choosing ϵ sufficiently small, they obtain the following corollaries:

1. Because $\mathsf{Enc}_{\mathsf{pr}}(\boldsymbol{x})$ and $\mathsf{Enc}_{\mathsf{pr}}(\boldsymbol{y})$ are resilient to parity leakage, their additive sharings are 2-BCLSSs of input $\boldsymbol{x}$ and output $\boldsymbol{y}$ respectively.
2. Because C_{pr} is a parity-resilient circuit, the additive sharing of internal wire values

$$(R, R \oplus W[C_{\mathsf{pr}}])$$

is also resilient to BCL.

However, the same method is difficult to apply directly to the case of 2-party LTMPC, even when starting with a *parity-tolerant circuit* C_{pr}. The main technical challenge is to reduce an arbitrary BCL of $(R, R \oplus W[C_{\mathsf{pr}}])$ to some BCL of (some partition of) the function input and output. Indeed, the only connection between parity leakage and BCL we know is from the small-biased masking lemma, which only tells us the indistinguishability between two distributions. It seems that the above reduction may require a stronger connection. As an evidence, the subsequent work [IS25] only goes as far as constructing a leakage-resilient computation against continuous leakage from a parity-tolerant circuit.

Strong Parity-Tolerant Circuits. The recent breakthrough by Ma and Song [MS26] presents a starting point for overcoming this barrier. Specifically, the authors propose a stronger notion of leakage-tolerant circuits in the sense of supporting leakage-oblivious simulation (called *strong LTCs*), and construct strong LTCs against AC^0-leakage as well as parity leakage with an exponentially small security loss. Their definition can be viewed as an analogy to leakage-tolerant computation protocols in the circuit setting. The simulator $\mathcal{S}$ of a strong LTC C outputs an $\mathsf{NC}^0[1]$ function T, independently of the leakage function, mapping the input-output pair $(\boldsymbol{x}, \boldsymbol{y} \leftarrow f(\boldsymbol{x}))$ to wire values of C, so that any leakage function $L \in \mathcal{L}$ applied on wire values of C can be simulated by $L \circ T(\boldsymbol{x}, \boldsymbol{y})$.

Our Approach. Given a strong parity-LTC C_{pr} of the functionality $\mathcal{F}$ in advance, we present a generic n-party LTMPC protocol for $\mathcal{F}$ in the $\mathcal{F}_{\mathsf{Mult}}$-hybrid model, where $\mathcal{F}_{\mathsf{Mult}}$ is an ideal leakage-tolerant abstraction of multiplication over additive sharings. Concretely, on input additive secret sharings $\langle\alpha\rangle_n = (\langle\alpha\rangle_{n,1}, \ldots, \langle\alpha\rangle_{n,n})$ and $\langle\beta\rangle_n = (\langle\beta\rangle_{n,1}, \ldots, \langle\beta\rangle_{n,n})$, $\mathcal{F}_{\mathsf{Mult}}$ uniformly samples a random additive secret sharing $\langle\gamma\rangle_n = \langle\alpha \cdot \beta\rangle_n = (\langle\gamma\rangle_{n,1}, \ldots, \langle\gamma\rangle_{n,n})$ conditioned on $\sum_{i=1}^{n}\langle\gamma\rangle_{n,i} = \alpha \cdot \beta$. Our protocol π_{MLTMPC} proceeds in the following way:

- Each party P_i distributes a random additive sharing of its input $\boldsymbol{x}_i$.
- The parties locally sample random sharings of the internal randomness of C_{pr}. Then they jointly compute an additive sharing of each wire in C_{pr} sequentially. Specifically, let g be a gate with

input wires α, β and output wire γ. If g is an addition gate, each party locally adds their shares of α, β to obtain share of γ; and if g is a multiplication gate, the parties invoke the ideal functionality $\mathcal{F}_{\mathsf{Mult}}$ to compute a sharing of γ.

- Each party P_i collects the shares from other parties and reconstructs its output $\boldsymbol{y}_i$.

We claim that our protocol π_{MLTMPC} is leakage-tolerant against at most $n-1$ semi-honest corruptions. The analysis can be divided into two cases:

1. When the corruption threshold $t \leq n-2$, we can easily generalize the small-biased masking lemma and show that in this case, the additive sharing of $\mathsf{NC}^0[1]$ function $T(\boldsymbol{x}, \boldsymbol{y})$ produces a distribution indistinguishable from the sharings of actual wire values under BCL. Furthermore, the additive sharing of each element $w = u^w \cdot a^w + v^w$ of T (say, $u^w, v^w \in \mathbb{F}_2$ are constants and a^w is an input or output variable of P_1) can be obliviously simulated: the simulator $\mathcal{S}$ prepares a random sharing $(v_1, \ldots, v_n)$ of v^w in advance, then P_1 uses the share $v_1 + u^w \cdot a^w$ and each other party uses its share of v^w in the simulation process.
2. When the corruption threshold $t = n-1$, the small-biased masking lemma falls outside the scope of its applicability, since the adversary can hand its shares to the unique honest party and use 1-bit leakage query to distinguish $T(\boldsymbol{x}, \boldsymbol{y})$ from the real-life distribution. Despite all this, since our construction can be viewed as an implementation of the semi-honest GMW protocol [Gol09, GMW87] over C_{pr}, we can show that the leakage-oblivious simulation property of π_{MLTMPC} directly follows from the adaptive security of the GMW protocol.

2.2 Computationally Secure LTMPCs.

Building upon the framework of Section 2.1, we show how to construct computationally secure semi-honest LTMPC protocols in the plain model by instantiating the ideal multiplication functionality $\mathcal{F}_{\mathsf{Mult}}$. Specifically, we reduce $\mathcal{F}_{\mathsf{Mult}}$ to the standard semi-honest 2-choose-1 leakage-tolerant oblivious transfer (LTOT) functionality. It is known that the semi-honest LTOT can be realized via augmented non-committing encryption [BCH12, GIM+16, CLOS02], while the latter primitive can be derived from the computational hardness of various assumptions: CDH, RSA, DDH, LWE or factoring Blum integers [CDSMW09].

2-Party LTMPC. The simplest case is to implement the 2-party $\mathcal{F}_{\mathsf{Mult}}$ functionality. For this, we make a slight modification to Lemma 1 of [GIM+16]. Assuming the additive sharings of α, β are (α_1, α_2) and (β_1, β_2), respectively, we seek to compute a random sharing of the output wire γ, say (γ_1, γ_2). In a nutshell, our protocol works as follows: the first party locally samples its output share γ_1, then two parties carry out a 4-choose-1 LTOT to let the second party learn its share $\gamma_2 = (\alpha_1 + \alpha_2) \cdot (\beta_1 + \beta_2) + \gamma_1$, while the use of 4-choose-1 LTOT can be simply replaced by 4 repetitions of 2-choose-1 LTOT.

Constant Party LTMPC. We then present constructions of multi-party LTMPC protocols. A natural approach is to consider a recursive reduction from n-LTMPC to $(n-1)$-LTMPC. For this, our first attempt incorporates the idea used in constructing the 2-LTMPC protocol. Specifically, we let the first $n-1$ parties $P_1 \sim P_{n-1}$ emulate the role of P_1 in 2-LTMPC and compute

$$\gamma(u, v) := \left(u + \sum_{i=1}^{n-1} \alpha_i\right) \cdot \left(v + \sum_{i=1}^{n-1} \beta_i\right) + \sum_{i=1}^{n-1} \gamma_i$$

where α_i, β_i are the input shares of P_i and γ_i is the output share of P_i. Because each party only knows its individual input and output, we distribute each $\gamma(u, v)$ using an $(n-1)$-BCLSS $\langle \gamma(u,v) \rangle_{n-1}^{\mathsf{BCL}}$ among $P_1, \ldots, P_{n-1}$ (henceforth denoted as $(\gamma(u,v)_1, \ldots, \gamma(u,v)_{n-1})$). After that, party P_n runs a 4-choose-1 LTOT with each P_i ($i \in \{1, \ldots, n-1\}$) to obtain the share $\gamma(\alpha_n, \beta_n)_i$, and recovers $\gamma_n = \gamma(\alpha_n, \beta_n)$ at the end of the protocol. This approach fails due to the leakage-oblivious simulation requirement: since every share $\gamma(\alpha_n, \beta_n)_i$ appears twice in the views of both P_i and P_n, in order to maintain the consistency of the views of different parties, the simulator must sample the share using common information between P_i and P_n, excluding any private inputs and outputs. Consequently, the simulation of the entire sharing $\langle \gamma(\alpha_n, \beta_n) \rangle_{n-1}$ can be finished independently of $(\langle \alpha \rangle_n, \langle \beta \rangle_n, \langle \gamma \rangle_n)$. This seems to be an impossible task when no party is corrupted.

To resolve this issue, we alter the construction to let the party P_n learn exactly one secret share $\gamma(\alpha_n, \beta_n)_{n-1}$. The side effect of this modification is that P_n can no longer reconstruct γ_n on its own. To compensate, we have $P_1, \ldots, P_{n-2}$ and P_n run another secure $(n-1)$-LTMPC protocol to finish the decoding process. In this way, each multiplication gate is instantiated by at most 2 invocations of $(n-1)$-LTMPC and 1 invocation of 2-LTMPC. Recursively implementing this algorithm, we immediately obtain efficient n-LTMPC protocols for every constant n.

Generic Solution. Our next goal is to obtain a generic n-party semi-honest LTMPC protocol using the party virtualization technique [Bra87]. At a high level, our construction develops the method of Applebaum and Kachlon [AK25], which amplifies constant-party leakage-resilient MPC protocols to multi-party leakage-resilient protocols.

We begin by describing a simplified version of our protocol in a weaker *independent leakage* model. In this model, the adversary is required to determine all leakage functions for honest parties before observing any information about the view of corrupted parties or the results of any leakage queries. Suppose that the number of parties n is even. To compute an n-party function $\boldsymbol{y} \leftarrow \mathcal{F}(\boldsymbol{x})$, we divide the n parties $P_1, \ldots, P_n$ into $n/2$ virtual parties such that the i-th virtual party consists of $\{P_{2i-1}, P_{2i}\}$ (denoted by $V_{2i-1,2i}$). In this way, the functionality $\mathcal{F}$ is equivalent to an $(n/2)$-party functionality $\mathcal{F}'$ over virtual parties, where the input (resp. output) of $V_{2i-1,2i}$ is set to be $\boldsymbol{x}_{2i-1} \parallel \boldsymbol{x}_{2i}$ (resp. $\boldsymbol{y}_{2i-1} \parallel \boldsymbol{y}_{2i}$). We compile a standard (non-leakage-tolerant) semi-honest $n/2$-party MPC protocol π_{Ext} computing $\mathcal{F}'$ which is secure against $(n/2-1)$ corruptions in the OT-hybrid model, into an n-LTMPC protocol by instantiating each point-to-point communication channel and each OT call with a 4-LTMPC sub-protocol. For more details, the view of every virtual party $V_{2i-1,2i}$ is shared between $\{P_{2i-1}, P_{2i}\}$ using 2-BCLSS, so we can explain each message delivery or OT call of π_{Ext} between sender $V_{2i-1,2i}$ and receiver $V_{2j-1,2j}$ as a 4-LTMPC protocol between $\{P_{2i-1}, P_{2i}, P_{2j-1}, P_{2j}\}$.

The independent leakage model provides us with a stronger security guarantee than the leaky-UC model: we can show that even if one of P_{2i-1} and P_{2i} is corrupted, the security of 2-BCLSS of a virtual party $V_{2i-1,2i}$ still holds unconditionally. Therefore, we can think of $V_{2i-1,2i}$ as a corrupted party in π_{Ext} if both parties P_{2i-1} and P_{2i} are corrupted in the LTMPC, and treat the 2-BCLSS of the transcript between honest virtual parties as one of all-0 strings. After this modification, the view of the adversary simply becomes a random 2-BCLSS of the view of corrupted parties in π_{Ext}, hence the security of our protocol follows from the security of π_{Ext}.

Unfortunately, this straightforward construction cannot be extended to the leaky-UC model, as the 2-BCLSS of $V_{2i-1,2i}$ does not preserve any privacy against the adversary when one of the parties is corrupted. To resolve this issue, we allocate $m = n(n-1)/2$ virtual parties: $V_{i,j} = (P_i, P_j)$ for every unordered pair (i, j) (for descriptive convenience, we also denote $V_{j,i} = V_{i,j}$). Analogous to the independent leakage case, we compile a semi-honest m-party GMW protocol π_{Ext} into an n-

LTMPC protocol, and this time we view each $V_{i,j}$ as a corrupted virtual party in π_{Ext} if at least one of P_i, P_j is corrupted. To analyze the new protocol, we observe that the security of our construction against $(n-1)$ corruptions follows from the adaptive security of π_{Ext} as in the proof of semi-honest case. Consequently, we only need to consider the case of $t \leq n-2$ corruptions. Under this premise, every honest party P_i belongs to at least one honest virtual party $V_{i,j}$ in which the corresponding 2-BCLSS is private against the adversary. Thus, we are able to prove security using an approach similar to the independent leakage model.

However, some subtleties remain in the above framework. Since we allow an actual party P_i to belong to multiple virtual parties $V_{i,j}$, the inputs and outputs of virtual parties cannot be naturally defined as in the independent leakage model. To solve this problem, we ask each party P_i to sample a length-$(n-1)$ additive sharing $\langle \boldsymbol{x}_i \rangle_{n-1} := (\boldsymbol{x}_{i,1}, \ldots, \boldsymbol{x}_{i,i-1}, \boldsymbol{x}_{i,i+1}, \ldots, \boldsymbol{x}_{i,n})$ of its private input $\boldsymbol{x}_i$, and define the input of virtual party $V_{i,j}$ as $\boldsymbol{x}_{i,j} \parallel \boldsymbol{x}_{j,i}$. Symmetrically, we let the protocol π_{Ext} distribute a length-$(n-1)$ additive sharing $\langle \boldsymbol{y}_i \rangle_{n-1}$ of the private output $\boldsymbol{y}_i$ among virtual parties $\{V_{i,1}, \ldots, V_{i,i-1}, V_{i,i+1}, \ldots, V_{i,n}\}$, enabling P_i to reconstruct the output $\boldsymbol{y}_i$. We can verify that this approach satisfies the leakage-oblivious simulation requirement of π_{Ext}: for a corrupted P_i, the simulator simply takes a random sharing of $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ as the inputs and outputs of the corresponding virtual parties; for an honest P_i, let V_{i,i^*} be an honest virtual party containing P_i. The simulator samples random strings as the inputs and outputs of $V_{i,j}$ $(j \neq i, i^*)$, while the input and output of V_{i,i^*} can be determined in the state translation function T_i according to $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$.

Handling Malicious Corruptions. We also discuss how to obtain multi-party LTMPC protocols in the presence of a dishonest majority and maliciously corrupted parties. The simplest and most practical approach is to compile the semi-honest LTMPC protocols into one with full security through the GMW compiler [GMW87, CLOS02]. Recall that the GMW compiler forces the corrupted parties to behave honestly by applying a leakage-tolerant commit-and-prove functionality, which can be implicitly derived from [BCH12] in the common reference string (CRS) model. We thus almost reach the final goal.

However, as observed in the seminal work [BGJ+13], we are unable to realize a secure 2-LTMPC protocol of the coin-tossing functionality (more generally, n-LTMPC that tolerates up to $n-1$ malicious corruptions). For completeness, we defer a formal proof of this barrier to Appendix A. As a result, the GMW compiler fails to enforce that the corrupted party chooses "good" randomness. Indeed, a rushing adversary may tamper with its share of the function randomness to bias the function outcome after seeing the shares of honest parties. As a compromise, we therefore relax the security notion to only consider deterministic functions when considering $t = n-1$ corruptions.

Even with the above relaxation, deriving the maliciously secure LTMPC remains non-trivial. For the special case of 2-party computation, we adopt the following approach of [BGJ+13]: first, we build an LTMPC protocol against a malicious adversary that follows the protocol execution as a semi-honest adversary but may adaptively choose its input and randomness depending on the leakage from the honest party (formalized by the notion of *semi-malicious* security [AJLA+12, BGJ+13]); then we compile it to a fully secure protocol via commit-and-prove. The authors of [BGJ+13] observed that the semi-honest GMW protocol (in the OT-hybrid model) is also semi-maliciously secure, while our GMW-based protocol π_{MLTMPC} inherits this property as well. Combining with any semi-malicious LTOT, we then fully resolve the 2-party case [3].

[3] We note that the result of [BGJ+13] does not immediately imply a 2-party LTMPC since their security notion allows the adversary to obtain joint leakage of the ideal states of multiple honest parties. Such leakage clearly cannot be simulated by individual leakages of each honest party as the leakage-tolerant computation model restricts. In particular, we cannot derive a 2-party LTMPC with no corruption from their result.

We then explore how to construct generic n-party malicious LTMPC protocols. In a similar vein to our semi-honest construction, we compile a non-leakage-tolerant malicious MPC protocol π_{Ext} into an n-party LTMPC protocol via 2-party LTMPC. However, since we do not construct a constant party malicious LTMPC, we cannot realize π_{Ext} in the OT-hybrid model as we did in the semi-honest case. A compromise solution is to start with a malicious MPC protocol in broadcast channel model, so that the only remaining interactive step between virtual parties is message broadcast. To broadcast a message m from a virtual party $V_{i,j}$, the parties (P_i, P_j) can jointly run a *semi-honest* 2-LTMPC to compute this message and broadcast it. Here, we carry out the internal computation using only semi-honest 2-LTMPC, because the external protocol π_{Ext} allows a corrupted virtual party to behave arbitrarily maliciously, while the truthfulness of a virtual party $V_{i,j}$ naturally implies that both underlying parties P_i and P_j are honest.

Unfortunately, the above method is vulnerable to $t = n-1$ corruptions. In this case, none of the virtual parties is honest, thus the malicious security of π_{Ext} brings neither output correctness nor privacy for the unique honest party. To this end, we also employ the n actual parties $P_1, \ldots, P_n$ as participants of π_{Ext}. The input $\boldsymbol{x}_i$ is now shared between $\{V_{i,j}\}_{j \neq i}$ as well as $V_i = \{P_i\}$ itself. This does not affect the analysis of $t \leq n-2$ corruptions because we can simply mark every $V_i = \{P_i\}$ as corrupted. We now design a semi-malicious m-party MPC protocol (for $m = n(n+1)/2$) that is leakage-tolerance when considering exactly $m-1$ corruptions, then compile it into π_{Ext} using the commit-and-prove technique. Such a protocol π_{Ext} is not difficult to construct, and we provide the details in Section 6.2.

2.3 Unconditionally Secure LTMPC

Another natural direction is to study the plausible constructions of unconditionally secure LTMPC in the honest majority setting. Somewhat surprisingly, this goal is actually *impossible* to achieve. Concretely, we show that for every $n \geq 2$, there always exists some functionality $\mathcal{F}$ that cannot be realized by n-LTMPC protocols with leakage-oblivious simulation in the plain model. Our impossibility result already holds even in a very weak setting: the leakage bound can be as small as $\ell = 2$, and no corruption is required. We refer readers to Section 7.1 for more details.

$(2,0)$-LTMPC. This impossibility result sharply rules out general-purpose unconditionally secure LTMPC protocols with leakage-oblivious simulation in the plain model. To circumvent the aforementioned barrier, we additionally assume that any honest party can securely erase its local view. Erasure is widely applied in the literature of non-leakage-tolerant adaptive MPC [BH92, Lin09, IPS09], and we observe that it likewise plays a crucial role in the construction of input-independent preprocessing LTMPC [BDSL14].

Our starting point is the generic compiler from simulatable public-key encryption (PKE) to semi-honest OT [CDSMW09]. In a nutshell, a simulatable PKE has the property that the public key pk and ciphertext e of a random plaintext $\boldsymbol{b}$ can be obliviously generated [4]. Surprisingly, assuming that (1) both the sender and receiver are honest; (2) the adversary's ability is limited to making leakage queries, the leakage-resilient one-time pad (LROTP) serves as an unconditionally secure simulatable PKE scheme in the erasure model. An LROTP is merely an encryption scheme $c \leftarrow \mathsf{Enc}(\mathsf{key}, b)$, given by the two-source inner-product extractor $\langle \mathsf{key}, c \rangle = b$. It was shown in [GR12] that a random LROTP of plaintext b is resilient to leakage queries. Concretely, let L be a bounded communication leakage attack independently applied to key and c, the plaintext bit b is statistically hidden against L.

[4] In the actual definition, we also require that the oblivious generation algorithms are invertible given a trapdoor, whereas the invertibility is trivial in our scheme.

For more details, we discuss how to obliviously encrypt a uniformly random bit. The receiver sends a random secret key key to the sender. Once the transmission is completed, both parties erase their local information about key. Then the sender sends a uniformly random c to the receiver. It is clear that $\langle \mathsf{key}, c \rangle$ is a random bit b. However, the security of LROTP ensures that the adversary cannot extract b with leakage queries: the views of both parties are determined only by key before erasure and determined only by c after that moment. As a result, the adversary can only query leakage from key and c separately, preserving the security of LROTP. Substituting this unconditionally secure simulatable PKE scheme into the construction of [CDSMW09], we then have a semi-honest 2-party LTOT scheme with no corruption. Combining this with the compiler of semi-honest 2-LTMPC, we immediately obtain a semi-honest $(2, 0)$-LTMPC.

Attempt to Generic Case. Leveraging the $(2, 0)$-LTMPC as a building block, our next goal is to design a generic unconditionally secure semi-honest LTMPC protocol in the erasure model. A first attempt is to merely use the party virtualization technique again: for each pair of parties (P_i, P_j), allocate a virtual party $V_{i,j}$ and use the $(2, 0)$-LTMPC protocol between (P_i, P_j) to emulate the internal computation of $V_{i,j}$; then we employ a standard statistically secure MPC protocol π_{Ext} (e.g. [BOGW88, CCD88]) over $m = n(n-1)/2$ virtual parties to implement the functionality $\mathcal{F}$. A virtual party $V_{i,j}$ is viewed as honest if and only if both components P_i and P_j are honest. The communication between two virtual parties can be trivially instantiated by pairwise sending a 2-BCLSS of the message. However, this strategy is effective only when the corruption threshold t does not exceed $\left(1 - \frac{\sqrt{2}}{2}\right) n$ asymptotically because any statistically secure protocol can tolerate at most $(m-1)/2$ corruptions in general [5].

To resolve this issue, we modify the construction to simply employ a statistically secure protocol π_{Ext} over n virtual parties $V_1, \ldots, V_n$, where each V_i is jointly emulated by all n actual parties $P_1, \ldots, P_n$ and has a representative party ("dealer") P_i. We think of each virtual party V_i as corrupted if the dealer P_i is. From now on, the problem becomes securely emulating the internal computation of each virtual party. The major merit of this approach is that the external protocol π_{Ext} only requires security to hold for an honest virtual party V_i. In other words, the protocol for emulating V_i suffices to be secure only when the dealer P_i is honest. To this end, we make use of the standard Beaver triples preprocessing MPC [Bea91] as such a protocol, except that we compile the circuit representing V_i with our LTMPC protocol in the $\mathcal{F}_{\mathsf{Mult}}$-hybrid model described in Section 2.1. Specifically, assuming there exists an ideal preprocessing functionality outputting n-party Beaver triples $(\langle a \rangle_n, \langle b \rangle_n, \langle c \rangle_n = \langle ab \rangle_n)$, we can implement the multiplication function $\mathcal{F}_{\mathsf{Mult}}$ of input wires α, β and output wire γ in a usual way: each party opens $\langle \alpha \rangle_n - \langle a \rangle_n$ to get ϵ and opens $\langle \beta \rangle_n - \langle b \rangle_n$ to get δ, then outputs $\langle \gamma \rangle_n = \epsilon\delta + \epsilon\langle b \rangle_n + \delta\langle a \rangle_n + \langle c \rangle_n$.

The Preprocessing Phase. We now turn to the more challenging task of generating Beaver triples. In the standard MPC model, once we convince the parties that there exists an honest dealer, we can present a very simple preprocessing protocol: just let the dealer locally prepare all Beaver triples and distribute them to other parties. Unfortunately, this strategy clearly fails to resist leakage, because the adversary can gain joint leakage of different parties' shares from a single leakage query of the dealer who knows all parties' shares. To address this, we design a 2-source randomness extractor $\mathsf{Ext}(X, Z)$ that can extract random (single party) Beaver triples from a specific distribution D over X. In addition, the extractor is linear in X (even if X is not sampled

[5] Using a double-layer party virtualization protocol, we can indeed generalize this idea to attain security against $t \leq \frac{n}{2} - 1$ corruptions. Unfortunately, we can show that it is impossible to obtain an (n, t)-LTMPC protocol for $n = 2t + 1$ with only recursive party virtualization technique (in particular, $(3, 1)$-LTMPC).

from D), meaning that $\mathsf{Ext}(X_1 + X_2, Z) = \mathsf{Ext}(X_1, Z) + \mathsf{Ext}(X_2, Z)$. For more details, the honest dealer P_i prepares a random n-party additive sharing $\langle X \rangle_n = (X_1, \ldots, X_n)$ for some $X \xleftarrow{\$} D$ and sends the shares to the other parties. Then it erases the randomness sampled so far except for its own share X_i. Thereafter, it samples a random seed Z and helps each $P_j \neq P_i$ to extract $(a_j, b_j, c_j) \leftarrow \mathsf{Ext}(X_j, Z)$. By the linearity of the extractor, $(\langle a \rangle_n, \langle b \rangle_n, \langle c \rangle_n) = ((a_j)_{j\in[n]}, (b_j)_{j\in[n]}, (c_j)_{j\in[n]})$ is an additive sharing of $(a, b, c) \leftarrow \mathsf{Ext}(X, Z)$ where the latter is a random Beaver triple.

For the extraction process, the dealer P_i runs a $(2, 0)$-LTMPC protocol with every $P_j \neq P_i$ to evaluate $(a_j, b_j, c_j) \leftarrow \mathsf{Ext}(X_j, Z)$, where P_i provides the input Z and P_j provides the input X_j. A seemingly severe issue appears when P_j is a corrupted party; in this case the $(2, 0)$-LTMPC protocol does not provide any security guarantee and may leak Z entirely. Fortunately, we observe that our 2-round $(2, 0)$-LTOT protocol maintains leakage tolerance for the receiver even against a corrupted sender, since the receiver always sends random secret keys no matter what the input bit b is. As a result, the LTMPC protocol implemented in this way preserves leakage tolerance for the party that plays the role of the receiver of LTOT, even if another party is corrupted. As a consequence, the input Z of P_i is only subject to bounded communication leakage (rather than being revealed to the adversary entirely) if we let the dealer P_i be the receiver of LTOT.

Handling Malicious Corruptions. Subsequently, we provide a construction that enhances the unconditionally secure semi-honest LTMPC to be secure against malicious adversaries. A direct approach is to replace the external MPC π_{Ext} with a fully secure MPC scheme in the honest majority setting (for instance, Rabin and Ben-Or [RBO89]). This attempt does not go through as directly as it seems. While we do not need to consider the internal computation of a corrupted virtual party V_i, the issue is to protect the internal computation of an honest V_i: our dealer-respect LTMPC with an honest dealer P_i does not provide a security guarantee as long as the corrupted parties behave dishonestly.

Designating an honest party P_i as the dealer, we propose a maliciously secure dealer-respect LTMPC. In principle, the technique used here is standard: we simply attach a leakage-resilient message authentication code (MAC) to every secret sharing, and require the corrupted party to provide correct share in every multiplication gate. For more details, we employ the information-theoretic MAC scheme of Bendlin, Damgård, Orlandi and Zakarias [BDOZ11] (henceforth abbreviated as BDOZ MACs). Let $\mathbb{F}$ be a sufficiently large field. The BDOZ MAC of a message $x \in \mathbb{F}$ under a key $K = (\Delta, \zeta) \in \mathbb{F}^2$ can be written as

$$\mathsf{MAC}_K(x) := (x, \zeta + \Delta x).$$

For two keys $K_1 = (\Delta, \zeta_1)$ and $K_2 = (\Delta, \zeta_2)$ sharing the same global secret key Δ, we define $K_1 + K_2 = (\Delta, \zeta_1 + \zeta_2)$. Then, two MACs $\mathsf{MAC}_{K_1}(x_1)$ and $\mathsf{MAC}_{K_2}(x_2)$ can be added up as $\mathsf{MAC}_{K_1}(x_1) + \mathsf{MAC}_{K_2}(x_2) = \mathsf{MAC}_{K_1+K_2}(x_1 + x_2)$.

Given a secret sharing $\langle \alpha \rangle_n = (\alpha_1, \ldots, \alpha_n)$, the BDOZ authenticated sharing (with dealer P_i) is defined as $[\alpha]_n = (\langle \alpha \rangle, \{K_j\}_{j\neq i}, \{\mathsf{MAC}_{K_j}(\alpha_j)\}_{j\neq i})$, where the dealer P_i holds α_i and secret keys $\{K_j = (\Delta, \zeta_j)\}_{j\neq i}$, and each other party P_j holds $\mathsf{MAC}_{K_j}(\alpha_j)$ [6]. The security proof of BDOZ sharing follows from a simple observation: a malicious adversary who tampers with some corrupted party P_j's share α_j to another value α'_j can only convince the dealer P_i with probability $1/|\mathbb{F}|$, because in this way the adversary can guess the global secret key Δ as

$$(1, \Delta) = (\alpha'_j - \alpha_j)^{-1}(\mathsf{MAC}_{K_j}(\alpha'_j) - \mathsf{MAC}_{K_j}(\alpha_j)).$$

[6] In the original proposal of [BDOZ11], each ordered pair of parties (P_x, P_y) shares a pair of $(K, \mathsf{MAC}_K(\alpha_y))$ where P_x holds key K and P_y holds $\mathsf{MAC}_K(\alpha_y)$. Because in our scheme the dealer P_i is always assumed to be honest, we can simplify their construction to only let the dealer hold keys.

We observe that the security of BDOZ sharing is maintained even in the presence of leakage, as long as $\log|\mathbb{F}| \geq \ell + \omega(\log\kappa)$: obtaining an arbitrary ℓ-bit leakage of Δ only helps the adversary guess the key with an advantage no more than $2^{\ell}/|\mathbb{F}| = \mathsf{negl}(\kappa)$. This implies that BDOZ MACs are also leakage-resilient MACs. As a result, assuming an ideal preprocessing functionality of BDOZ triples (namely, authenticated Beaver triples), we obtain a dealer-respect LTMPC protocol by properly modifying the previous construction.

For the preprocessing phase, we simply modify the $(2,0)$-LTMPC to also output MACs $(\mathsf{MAC}_{\cdot}(a_j), \mathsf{MAC}_{\cdot}(b_j), \mathsf{MAC}_{\cdot}(c_j))$ after extracting $(a_j, b_j, c_j) \leftarrow \mathsf{Ext}(X_j, Z)$ for each party $P_j \neq P_i$. However, a subtlety arises when we consider the extraction of a corrupted P_j. Recall that in the semi-honest case, we argue the security of this step by relying on the leakage tolerance for the receiver against a semi-honest sender in our LTOT. Evidently, the same proof breaks down completely when considering a malicious adversary. Indeed, the adversary might deviate in the $(2,0)$-LTMPC to obtain arbitrary information about Z. To address this, we re-examine the construction of our $(2,0)$-LTMPC in a non-black-box way. The dealer P_i only sends three types of messages to the corrupted party P_j during the protocol execution: (1) an input share of Z; (2) the secret keys of the LROTPs in each LTOT; (3) the share of P_j's outputs. The first two types of messages are perfectly random regardless of P_j's (possibly dishonest) behaviors, while the length of the last type of message does not exceed $O(\log|\mathbb{F}|)$ bits, independently of the length of Z. Counting over all corrupted parties, the adversary gains at most $O(n\log|\mathbb{F}|)$-bit leakage of Z in this way. Hence, the preprocessing scheme remains secure if we choose Z to be sufficiently long, so that it retains high residual entropy after leakage.

3 Preliminaries

Notation. In this work, we consider the standard arithmetic circuit model over the finite field $\mathbb{F}_2$ unless otherwise stated. We use κ to denote the security parameter, and use $\mathsf{negl}(\kappa)$ to denote the functions that decay to 0 faster than $1/\kappa^c$ for every constant $c > 0$. For a finite set (resp. distribution) D, we let $x \xleftarrow{\$} D$ denote an element x that is sampled uniformly at random from D (resp. according to distribution D). We let U_n denote the uniform distribution over n-bit strings $\{s : s \in \mathbb{F}_2^n\}$. Furthermore, we use the notation $\langle a\rangle_n = (\langle a\rangle_{n,1}, \ldots, \langle a\rangle_{n,n})$ to denote a length-n additive sharing of a over $\mathbb{F}_2$.

3.1 Leakage-Tolerant Computation

Our leakage-tolerant computation results are presented in the leaky-UC model, following the approach of [BCH12,BDSL14]. Here we briefly introduce the leaky-UC model, which is an extension of the universally composable (UC) secure computation framework [Can01] that allows the adversary to obtain partial information about the honest parties' views via leakage execution.

UC Protocol Execution. We first review the basic protocol execution formulation in the UC security model [Can01]. In this work, we assume the adversary is *static* in the sense that it only corrupts parties before the protocol execution.

A protocol π consists of n parties $P_1, \ldots, P_n$ such that a subset C of parties are corrupted. The i-th party P_i is a *probabilistic polynomial-time (P.P.T.) interactive Turing machine (ITM)* that begins with an input tape $\boldsymbol{x}_i$ (initially empty), its random tape $\boldsymbol{r}_i$, and the security parameter κ. The environment $\mathcal{Z}$ is an ITM with input $\boldsymbol{z}$, random tape $\boldsymbol{r}_{\mathcal{Z}}$ and the security parameter κ. $\mathcal{Z}$

may append information to the input tapes of other ITMs (including the adversary) and read the outputs of their output tapes, at any time during the protocol.

In the real-world model, there exists a real-world adversary $\mathcal{A}$ that starts with its random tape $\boldsymbol{r}_{\mathcal{A}}$ and the security parameter κ. Both $\mathcal{Z}$ and $\mathcal{A}$ learn the identities of C.

Once a party P_i is activated, it follows the protocol unless it is corrupted, locally generates output on its output tape and delivers messages to another party P_j. $\mathcal{A}$ gains access to all the tapes of corrupted parties, and controls the behaviors of those corrupted parties if the adversary is malicious. In contrast, the corrupted parties follow the protocol as usual if the adversary is semi-honest.

Let $\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa, \boldsymbol{z})$ denote the random variable representing the output of $\mathcal{Z}$ in a real-world execution. Let $\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa, \boldsymbol{z})\}_{\kappa\in\mathbb{N},\boldsymbol{z}\in\{0,1\}^*}$.

Definition 3.1 (Protocol Emulation)**.** Let π and ϕ be protocols. We say that π t-privately (resp. t-securely) UC-emulates ϕ if, for any real-world semi-honest (resp. malicious) adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that for any corruption set C of size at most t, no environment $\mathcal{Z}$ can distinguish the interaction with π and $\mathcal{A}$ from one with ϕ and $\mathcal{S}$.

According to the level of indistinguishability between ensembles $\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\mathsf{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}$, we provide two security notions. Formally, the emulation is computationally secure if for all P.P.T. $\mathcal{A}$, there exists a P.P.T. $\mathcal{S}$ such that for all P.P.T. $\mathcal{Z}$,

$$\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}} \approx_C \mathsf{EXEC}_{\phi,\mathcal{S},\mathcal{Z}},$$

and is unconditionally secure if for all $\mathcal{A}$, there exists $\mathcal{S}$ such that for all $\mathcal{Z}$,

$$\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}} \approx_S \mathsf{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}.$$

Ideal Functionalities and Ideal Protocols. In the ideal process, there exists an *ideal functionality* $\mathcal{F}$, as well as an adversary (simulator) $\mathcal{S}$ that starts with random tape $\boldsymbol{r}_{\mathcal{S}}$ and the security parameter κ. All of $\mathcal{Z}$, $\mathcal{S}$ and $\mathcal{F}$ learn the identities of C.

The ideal functionality $\mathcal{F}$ corresponds to a special type of protocol called an *ideal protocol*. When the ideal protocol $I_{\mathcal{F}}$ of $\mathcal{F}$ receives messages from other parties, it follows the code of $\mathcal{F}$ to compute the ideal functionality output, and potentially sends messages to parties P_i and simulator $\mathcal{S}$.

Each party P_i proceeds in a dummy manner and no longer uses the random tape $\boldsymbol{r}_i$: it delivers every input received from the environment to $I_{\mathcal{F}}$ and copies the outgoing message from $I_{\mathcal{F}}$ to its own output tape. $\mathcal{S}$ gains access to all tapes of corrupted parties. If the adversary $\mathcal{S}$ is malicious, it controls the corrupted parties' actions and the messages sent to $I_{\mathcal{F}}$.

Definition 3.2 (Realizing Functionalities)**.** Let π be a protocol and $\mathcal{F}$ be an ideal functionality. We say that π t-privately (resp. t-securely) UC-realizes $\mathcal{F}$ if π t-privately (resp. t-securely) UC-emulates the corresponding ideal protocol $I_{\mathcal{F}}$.

Leakage Execution. In the leaky-UC model, the real-world adversary $\mathcal{A}$ can additionally perform a *leakage* execution on the *view* view_i of an honest party P_i (a.k.a. entire state, as called in [BCH12, BDSL14]) consisting of the input tape $\boldsymbol{x}_i$, the random tape $\boldsymbol{r}_i$ used so far, and the messages exchanged with other parties (namely, the transcript). The leakage execution is specified by the identity of party P_i, and a leakage function L on view_i. Once the adversary obtains the leakage result $L(\mathsf{view}_i)$, the environment $\mathcal{Z}$ learns the identity of P_i along with the number of leaked bits. An aggregator is used to account for the number of leaked bits from leakage executions [7]. In the

[7] To simplify the analysis, we define a single aggregator and a global leakage bound ℓ for all parties. This slightly differs from the definition of [BDSL14], which allows each party to independently leak at most ℓ bits.

ideal process, the leakage execution of an honest party P_i is applied only to an *ideal state* state_i corresponding to the ideal functionality $\mathcal{F}$, defined as the messages exchanged between P_i and $I_{\mathcal{F}}$.

We also consider protocols in the hybrid model, where a real protocol π may recursively invoke sub-protocols $\rho_1, \ldots, \rho_k$. In this case, the view of a party P_i also includes its ideal state of any subroutine ρ_j.

Leaky-UC Security. The notion of leaky-UC emulation is analogous to the notion of UC emulation, such that the environment cannot distinguish a real interaction from a simulated interaction, even with leakage executions. As in [BDSL14], most of the results in our work consider a restricted model, where the indistinguishability is only required to hold as long as the number of leaked bits does not exceed a priori bound ℓ.

Definition 3.3 (Leaky-UC Protocol Emulation)**.** Let π and ϕ be leaky protocols. We say that π t-privately (resp. t-securely) UC-emulates ϕ if, for any real-world semi-honest (resp. malicious) leaky-adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that for any corruption set C of size at most t, no environment $\mathcal{Z}$ can distinguish the interaction with π and $\mathcal{A}$ from one with ϕ and $\mathcal{S}$. The emulation is computationally secure if for all P.P.T. $\mathcal{A}$, there exists a P.P.T. $\mathcal{S}$ such that for all P.P.T. $\mathcal{Z}$,

$$\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}} \approx_C \mathsf{EXEC}_{\phi,\mathcal{S},\mathcal{Z}},$$

and is unconditionally secure if for all $\mathcal{A}$, there exists $\mathcal{S}$ such that for all $\mathcal{Z}$,

$$\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}} \approx_S \mathsf{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}.$$

We say that π UC-emulates ϕ under $\ell = \ell(\kappa)$ leakage, if the above holds only for every adversary that gains at most ℓ bits from the leakage execution.

As noted in [BDSL14], to prove leaky-UC security, it suffices to consider only a *dummy* adversary $\mathcal{D}$ that delivers all the messages sent by the parties to the environment and follows the instructions of the environment.

Leakage-Oblivious Simulation. To support universal composition, we follow [BCH12] and consider a stronger form of protocol emulation, where the simulator $\mathcal{S}$ is forced to *obliviously* handle the leakage execution without looking at the leakage function L. Concretely, an oblivious simulator $\mathcal{S}$ of a dummy adversary $\mathcal{D}$ invokes a separate subroutine $\widetilde{\mathcal{S}}$, designed for obliviously handling a leakage request of the form "applying a leakage function L to party P_i's view". To achieve this, $\widetilde{\mathcal{S}}$ outputs a *state translation function* $T = \{T_i\}$ that on input the ideal state state_i outputs a simulated view of party P_i. In this way, the leakage function L can be implemented by applying $L \circ T_i$ to state_i. We emphasize that the subroutine $\widetilde{\mathcal{S}}$ works independently of the leakage request: the only input of $\widetilde{\mathcal{S}}$ is the current state of $\mathcal{S}$, and the state of $\mathcal{S}$ is restored to the one before the leakage request after $\widetilde{\mathcal{S}}$ delivers the leakage result to the environment. Formally, we have the following definition:

Definition 3.4 (Leaky-UC Protocol with Oblivious Simulation)**.** Let π and ϕ be leaky protocols. We say that π t-privately (resp. t-securely) and strongly UC-emulates ϕ if there exists an oblivious simulator $\mathcal{S}$, such that for the semi-honest (resp. malicious) dummy adversary $\mathcal{D}$ and any corruption set C of size at most t, no environment $\mathcal{Z}$ can distinguish the interaction with π and $\mathcal{D}$ from one with ϕ and $\mathcal{S}$. The emulation is computationally secure if for all P.P.T. $\mathcal{A}$, there exists a P.P.T. $\mathcal{S}$ such that for all P.P.T. $\mathcal{Z}$,

$$\mathsf{EXEC}_{\pi,\mathcal{D},\mathcal{Z}} \approx_C \mathsf{EXEC}_{\phi,\mathcal{S},\mathcal{Z}},$$

and is unconditionally secure if for all $\mathcal{A}$, there exists $\mathcal{S}$ such that for all $\mathcal{Z}$,

$$\mathsf{EXEC}_{\pi,\mathcal{D},\mathcal{Z}} \approx_S \mathsf{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}.$$

We say that π strongly UC-emulates ϕ under $\ell = \ell(\kappa)$ leakage, if the above holds for every adversary that gains at most ℓ bits from the leakage execution.

Definition 3.5 (Leakage-Tolerant Computation)**.** Given an n-party ideal functionality $\mathcal{F}$, we say that $\mathcal{F}$ can be UC-realized by secure (n, t, ℓ) leakage-tolerant multi-party computation ((n, t, ℓ)-LTMPC for short) with semi-honest security (resp. malicious security), if there exists some protocol π that t-privately (resp. t-securely) and strongly UC-realizes $\mathcal{F}$ under ℓ leakage ($\ell = \infty$ if π strongly UC-realizes $\mathcal{F}$ even with an unbounded number of leakage bits). We may abbreviate the notation to (n, t)-LTMPC or n-LTMPC, when the parameters ℓ and t are clear in the context.

Remark 3.6. An additional issue arises when we consider the oblivious simulation requirement for randomized functionalities. Assume that the adversary corrupts $t = n - 1$ parties. The leakage-oblivious simulation property requires the simulator to create a state translation function which, on input the ideal state of the only honest party, outputs this honest party's view that is consistent with the simulated corrupted parties' view, and therefore determines the randomness used to generate the function output. Unfortunately, given only the ideal state of all corrupted parties, the simulator may not be able to recover the randomness in polynomial time. For example, consider the following functionality $\mathcal{F}_{\mathsf{RSA}}$ which randomly samples large primes p and q and outputs the product $n = pq$. Then, given only the ideal functionality output n, recovering the randomness p and q is as hard as factorization.

We note that the same issue has already occurred in the literature of adaptively secure UC computation without leakage. Here, we adopt the same patch as [CLOS02] that restricts the ideal functionality $\mathcal{F}$ to be *adaptively well-formed*, such that if $t = n - 1$ parties are corrupted, then the internal randomness of $\mathcal{F}$ is revealed to the oblivious simulation subroutine $\widetilde{\mathcal{S}}$ during the invocation of the leakage execution related to the remaining honest party.

Composition Theorem. [BCH12] shows that the notion of strong UC-emulation naturally preserves security under universal composition. Formally, let ρ be a protocol that invokes subroutine calls to ϕ. We denote by ρ^{ϕ} the protocols for which the subroutine calls are processed by ϕ, and by ρ^{π} the systems for which subroutine calls are computed using π. Then, we have:

Theorem 3.7 (UC-Composition with Leakage [BCH12])**.** *Let ρ, ϕ, π be leaky-protocols, such that ϕ strongly UC-emulates π. Then ρ^{ϕ} strongly UC-emulates ρ^{π}. Also, if ϕ strongly UC-emulates π under $\ell = \ell(\kappa)$ leakage, then ρ^{ϕ} strongly UC-emulates ρ^{π} under ℓ leakage.*

In this work, all of our protocols are designed under an ideal secure communication model as [BDSL14], in which each pair of parties (P_i, P_j) enjoys an authenticated, private point-to-point communication channel. Specifically, we present our protocols in the $\mathcal{F}_{\mathsf{SMT}}$-hybrid model, which can be instantiated using non-committing encryption and leakage-resilient MACs [BCG+11, BCH12].

Some of our results presented against malicious adversaries also make use of an ideal broadcast channel $\mathcal{F}_{\mathsf{BC}}$. Since we are content with achieving malicious security with abort, we can directly adopt the simple implementation of the broadcast channel from only point-to-point channels [GL02].

Function Evaluation. A weakened version (yet powerful enough) of the generic leaky-UC model is to only consider the ideal evaluation functionalities $\mathcal{F}$ [Can00, Can01]. Specifically, the ideal functionality $\mathcal{F}$ is defined as $\boldsymbol{y}_i \leftarrow \mathcal{F}_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, where $\boldsymbol{x}_i$ denotes the input of the i-th party and

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

Running with parties S, R and a simulator $\mathcal{S}$, $\mathcal{F}_{\mathsf{SMT}}$ works as follows:

- Upon receiving input (send, S, R, m) from party S, send $(\mathsf{send}, S, R, |m|)$ to $\mathcal{S}$. Once $\mathcal{S}$ acknowledges, send (sent, S, m) to R.

Figure 1: The Message Transmission Functionality

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

Running with parties $P_1, \ldots, P_n$ and a simulator $\mathcal{S}$, $\mathcal{F}_{\mathsf{BC}}$ works as follows:

- Upon receiving input $(\mathsf{broadcast}, P_i, m)$ from party P_i, send $(\mathsf{broadcast}, P_i, m)$ to both $P_1, \ldots, P_n$ and $\mathcal{S}$.

Figure 2: The Broadcast Functionality

$\boldsymbol{y}_i$ denotes the output of the i-th party. The ideal state of the i-th party is $(\boldsymbol{x}_i, \boldsymbol{y}_i)$. We note that the input $\boldsymbol{x}_i$ is determined by the environment $\mathcal{Z}$ before the protocol execution. $\mathcal{Z}$ is not allowed to append any value on the input tape of P_i once the protocol starts.

In this work, we focus solely on function evaluation, while most of the protocols can be naturally extended to support generic reactive functions.

3.2 Leakage-Tolerant Circuits

Definition 3.8 (Leakage-Resilient Encodings)**.** We say that a bit encoding scheme $(\mathsf{Enc}, \mathsf{Dec})$ is an $(\mathcal{L}, \epsilon)$-leakage-resilient encoding scheme, if it satisfies:

- **Correctness:** $\mathsf{Dec}(\mathsf{Enc}(b)) \equiv b$ for $b \in \{0, 1\}$.
- **Leakage-Resilience:** For all leakage functions $L \in \mathcal{L}$, we have

$$L(\mathsf{Enc}(0)) \approx_\epsilon L(\mathsf{Enc}(1)).$$

We may abbreviate the notation to $\mathcal{L}$-resilient encoding if $\epsilon = \mathsf{negl}(\kappa)$.

Definition 3.9 (Leakage-Tolerant Circuits)**.** Let f be a (possibly randomized) functionality. We say that a randomized circuit C is an $(\mathcal{L}, \epsilon)$-leakage-tolerant circuit implementation ($(\mathcal{L}, \epsilon)$-LTC for short) of f, if it satisfies the leakage-tolerance property: there exists a simulator $\mathcal{S}$, such that for any adversary $\mathcal{A}$ and input $\boldsymbol{x}$, the outputs of the following two experiments have a statistical distance of at most ϵ:

$$\mathsf{Real}_{\mathcal{A},\mathcal{L},\boldsymbol{x}} \approx_\epsilon \mathsf{Ideal}_{\mathcal{S},\mathcal{A},\mathcal{L},\boldsymbol{x}}.$$

We abbreviate the notation to $\mathcal{L}$-LTC if $\epsilon = \mathsf{negl}(\kappa)$.

Definition 3.10 (Strong Leakage-Tolerant Circuits)**.** Let f be a (possibly randomized) functionality. For a leakage class L that is closed under restriction and negation (i.e., $\mathcal{L} \circ \mathsf{NC}^0[1] \subseteq \mathcal{L}$), a circuit C is a strong $(\mathcal{L}, \epsilon)$-leakage-tolerant circuit (strong $(\mathcal{L}, \epsilon)$-LTC for short) implementation of f if it

Experiment $\mathsf{Real}_{\mathcal{A},\mathcal{L},\boldsymbol{x}}$

- Run C on input $\boldsymbol{x}$, obtain wire values W and output $\boldsymbol{y}$.
- $\boldsymbol{y} \to \mathcal{A}$.
- $\mathcal{A}$ outputs a leakage function $L \in \mathcal{L}$.
- $L(W) \to \mathcal{A}$.
- Output $(\mathsf{view}_{\mathcal{A}}, \boldsymbol{y})$.

Experiment $\mathsf{Ideal}_{\mathcal{S},\mathcal{A},\mathcal{L},\boldsymbol{x}}$

- Compute $\boldsymbol{y} \leftarrow f(\boldsymbol{x})$.
- $\boldsymbol{y} \to \mathcal{A}$.
- $\mathcal{A}$ outputs a leakage function $L \in \mathcal{L}$.
- On input L, $\mathcal{S}$ outputs a leakage function $L' \in \mathcal{L}$.
- $L'(\boldsymbol{x}, \boldsymbol{y}) \to \mathcal{A}$.
- Output $(\mathsf{view}_{\mathcal{A}}, \boldsymbol{y})$.

Figure 3: Leakage-Tolerant Circuit Experiments

satisfies the strong leakage-tolerance property. Specifically, there exists a simulator $\mathcal{S}$, such that for any adversary $\mathcal{A}$ and input $\boldsymbol{x}$, the outputs of the following two experiments are ϵ-indistinguishable:

$$\mathsf{Real}_{\mathcal{A},\mathcal{L},\boldsymbol{x}} \approx_{\epsilon} \mathsf{Ideal}_{\mathcal{S},\mathcal{A},\mathcal{L},\boldsymbol{x}}.$$

Similarly, we abbreviate the notation to strong $\mathcal{L}$-LTC if $\epsilon = \mathsf{negl}(\kappa)$.

Experiment $\mathsf{Real}_{\mathcal{A},\mathcal{L},\boldsymbol{x}}$

- Run C on input $\boldsymbol{x}$, obtain wire values W and output $\boldsymbol{y}$.
- $\boldsymbol{y} \to \mathcal{A}$.
- $\mathcal{A}$ outputs a leakage function $L \in \mathcal{L}$.
- $L(W) \to \mathcal{A}$.
- Output $(\mathsf{view}_{\mathcal{A}}, \boldsymbol{y})$.

Experiment $\mathsf{Ideal}_{\mathcal{S},\mathcal{A},\mathcal{L},\boldsymbol{x}}$

- Compute $\boldsymbol{y} \leftarrow f(\boldsymbol{x})$.
- $\boldsymbol{y} \to \mathcal{A}$.
- $\mathcal{S}$ outputs an $\mathsf{NC}^0[1]$ function T.
- $\mathcal{A}$ outputs a leakage function $L \in \mathcal{L}$.
- $L \circ T(\boldsymbol{x}, \boldsymbol{y}) \to \mathcal{A}$.
- Output $(\mathsf{view}_{\mathcal{A}}, \boldsymbol{y})$.

Figure 4: Strong Leakage-Tolerant Circuit Experiments

3.3 Bounded Communication Leakage

Given an input sequence $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ of n parties in advance, we use bounded communication leakage to characterize the information gained from a leaky-UC adversary:

Definition 3.11 (Bounded Communication Leakage)**.** Let $\ell = \ell(\kappa)$ be a leakage bound. On an input sequence $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \in (\{0,1\}^*)^n$ given in advance, a (n, t, ℓ)-leaky-UC adversary $\mathcal{A}$ can corrupt up to t parties and make up to ℓ-bit leakage queries on the inputs of honest parties. We say that L is a (possibly randomized) (n, t, ℓ)-bounded communication leakage function (henceforth denoted by $L \in \mathcal{L}_{\mathsf{BCL}}^{n,t,\ell}$), if there exists some (n, t, ℓ)-leaky-UC adversary $\mathcal{A}$ such that $L(\boldsymbol{x})$ is the output of $\mathcal{A}$ on input sequence $\boldsymbol{x}$.

Remark 3.12. Our definition of BCL leakage attack can be viewed as a natural extension of the corresponding notion in [GIM+16, GIW17]. While [GIM+16] defines BCL leakage as the transcript of some ℓ-bounded 2-party computation protocol π, we can show that this is merely the $(2, 0, \ell)$-BCL case in our definition.

On the one hand, the transcript of π can be drawn as the output of some $(2, 0, \ell)$-adversary $\mathcal{A}$ that emulates the protocol π in the leaky-UC model. On the other hand, let $\mathcal{A}$ be any $(2, 0, \ell)$-leaky-UC adversary acting on the input sequence $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2)$ and fix the random tape of $\mathcal{A}$. We can derive the output of $\mathcal{A}$ from the transcript of a corresponding ℓ-bounded protocol π as follows: each party of π simulates the internal state of $\mathcal{A}$; once $\mathcal{A}$ is willing to make a leakage query on $\boldsymbol{x}_i$, the i-th party of π simply outputs the corresponding leakage result. Finally, we can reveal the output of $\mathcal{A}$ from the transcript as it is just a deterministic function of the fixed randomness and leakage results.

Relation to Parity Resilience. Finally, we show that the parity resilience actually yields bounded communication leakage-resilience, by extending the small-biased masking lemma [GIM+16, IS25] to the n-party computation setting:

Lemma 3.13 (Generalized Small-Biased Masking Lemma [IS25])**.** *Let $\ell = \ell(\kappa)$ be a leakage bound. Let X and Y be two random variables of length k such that X, Y are ϵ-indistinguishable by parity leakage. Then the additive secret sharings $\langle X\rangle_n \in (\mathbb{F}_2^k)^n$ and $\langle Y\rangle_n \in (\mathbb{F}_2^k)^n$ are $2^{\ell/2}\epsilon$-indistinguishable by $\mathcal{L}_{\mathsf{BCL}}^{n,n-2,\ell}$ leakage.*

Lemma 3.13 is substantially the best possible result we can expect: suppose that the leaky-UC adversary can corrupt $t = n-1$ parties, then it can leverage a 1-bit leakage of the remaining honest party to distinguish any statistically far distributions (X, Y).

Proof. Lemma 2 of [IS25] has already established the $\mathcal{L}_{\mathsf{BCL}}^{2,0,\ell}$ case of Lemma 3.13. We thereby show a reduction from generic $(n, n-2, \ell)$-BCL to $(2, 0, \ell)$-BCL. Let $P_1, \ldots, P_n$ denote all parties. Suppose that there exists some $(n, n-2, \ell)$-leaky-UC adversary $\mathcal{A}$ that breaks the $\mathcal{L}_{\mathsf{BCL}}^{n,n-2,\ell}$ security game between $(\langle X\rangle_n, \langle Y\rangle_n)$ (W.L.O.G. assume that $\mathcal{A}$ corrupts parties $P_3, \ldots, P_n$). We then construct an adversary $\mathcal{A}'$ as follows:

- $\mathcal{A}'$ samples random shares $\boldsymbol{x}_3, \ldots, \boldsymbol{x}_n$.
- On an input sequence $\boldsymbol{x}' = (\boldsymbol{x}'_1, \boldsymbol{x}'_2)$, $\mathcal{A}'$ emulates the behavior of $\mathcal{A}$ on input sequence $\boldsymbol{x} = (\boldsymbol{x}_1 := \boldsymbol{x}'_1, \boldsymbol{x}_2 := \boldsymbol{x}'_2 - \sum_{i=3}^{n} \boldsymbol{x}_i, \boldsymbol{x}_3, \ldots, \boldsymbol{x}_n)$. Specifically, a leakage query to $\boldsymbol{x}_2$ can be translated to some leakage query to $\boldsymbol{x}'_2$, since $\mathcal{A}$ knows the values of $\boldsymbol{x}_3, \ldots, \boldsymbol{x}_n$.
- $\mathcal{A}'$ outputs the same values as $\mathcal{A}$.

We can see that depending on whether the input sequence is $\boldsymbol{x}' \xleftarrow{\$} \langle X\rangle_2$ or $\boldsymbol{x}' \xleftarrow{\$} \langle Y\rangle_2$, the sequence $\boldsymbol{x}$ generated by $\mathcal{A}$ will have the same distribution as $\langle X\rangle_n$ or $\langle Y\rangle_n$, respectively. Hence, the adversary $\mathcal{A}'$ breaks the $L_{\mathsf{BCL}}^{2,0,\ell}$ security game between $(\langle X\rangle_2, \langle Y\rangle_2)$, which causes a contradiction. □

An important application of Lemma 3.13 is to provide a secret sharing scheme resilient to bounded communication leakage from any parity-resilient encoding scheme. In the following, we present the formal definition:

Definition 3.14 (Bounded Communication Leakage-Resilient Secret Sharing)**.** Let $\ell = \ell(\kappa)$ be a leakage bound. We say that an n-party secret sharing scheme $(\mathsf{Enc}_{\mathsf{BCL}}, \mathsf{Dec}_{\mathsf{BCL}})$ is a (n, ℓ, ϵ)-bounded communication leakage-resilient secret sharing scheme (or (n, ℓ, ϵ)-BCLSS for short) if it satisfies the following conditions:

- On input a bit $b \in \{0,1\}$, $\mathsf{Enc}_{\mathsf{BCL}}(b)$ outputs secret sharing $\langle b \rangle_n^{\mathsf{BCL}} = (\langle b \rangle_{n,1}^{\mathsf{BCL}}, \ldots, \langle b \rangle_{n,n}^{\mathsf{BCL}})$.
- On input $(\langle b \rangle_{n,1}^{\mathsf{BCL}}, \ldots, \langle b \rangle_{n,n}^{\mathsf{BCL}})$, $\mathsf{Dec}_{\mathsf{BCL}}(\langle b \rangle_{n,1}^{\mathsf{BCL}}, \ldots, \langle b \rangle_{n,n}^{\mathsf{BCL}})$ outputs b with probability 1.
- $\boldsymbol{s} \leftarrow \mathsf{Enc}_{\mathsf{BCL}}(0)$ and $\boldsymbol{s} \leftarrow \mathsf{Enc}_{\mathsf{BCL}}(1)$ are ϵ-indistinguishable by $\mathcal{L}_{\mathsf{BCL}}^{n,n-2,\ell}$ leakage.

We ignore the symbol ϵ in notation if $\epsilon = \mathsf{negl}(\kappa)$.

An (n, ℓ, ϵ)-BCLSS is said to be perfectly simulatable if for any $i \in [n]$:

- Let $\boldsymbol{s}_{[n]/i} := (\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{i-1}, \boldsymbol{s}_{i+1}, \ldots, \boldsymbol{s}_n)$ be uniformly random shares, then the random variable of $\boldsymbol{s}_{[n]/i}$ is identical to $\langle b \rangle_{n,[n]/i}^{\mathsf{BCL}}$ for both $b \in \{0,1\}$.
- There exists an efficient algorithm SimSS, such that given any $\boldsymbol{s}_{[n]/i}$ and bit $b \in \{0,1\}$, $\mathsf{SimSS}(i, \boldsymbol{s}_{[n]/i}, b)$ outputs the share $\boldsymbol{s}_i$ of the i-th party satisfying

$$(\boldsymbol{s}_{[n]/i}, \boldsymbol{s}_i) \equiv \mathsf{Enc}_{\mathsf{BCL}}(b).$$

Moreover, we say that the BCLSS scheme satisfies the locally samplable property, if the random variable of n i.i.d. strings $\boldsymbol{s} = (\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n)$ is identical to a random share of a random secret bit (namely, $\boldsymbol{s} \equiv \{\mathsf{Enc}_{\mathsf{BCL}}(b) : b \xleftarrow{\$} \{0,1\}\}$) when every share $\boldsymbol{s}_i$ is independently and randomly chosen.

Corollary 3.15. *For all $n \geq 2$, security loss parameters ϵ and leakage bounds $\ell = \ell(\kappa)$, the n-party additive secret sharing of any parity-resilient encoding scheme $(\mathsf{Enc}_{\mathsf{pr}}, \mathsf{Dec}_{\mathsf{pr}})$ with security loss $\epsilon_{\mathsf{pr}} = 2^{-\ell/2}\epsilon$ is a perfectly simulatable (n, ℓ, ϵ)-BCLSS scheme:*

$$\mathsf{Enc}_{\mathsf{BCL}}(b) := \langle \mathsf{Enc}_{\mathsf{pr}}(b) \rangle_n.$$

Specifically, if we implement the underlying parity-resilient scheme by the inner-product encoding

$$\mathsf{Enc}_{\mathsf{pr}}(b) := (\boldsymbol{u}, \boldsymbol{v}, \langle \boldsymbol{u}, \boldsymbol{v} \rangle + b), \boldsymbol{u}, \boldsymbol{v} \xleftarrow{\$} \mathbb{F}_2^{\ell/2 + \log(1/\epsilon)},$$

the resulting BCLSS scheme is also locally samplable; in addition, each share has length $O(\ell + \log(1/\epsilon))$.

3.4 Entropy and Randomness Extractor

Definition 3.16. For a random variable A, the min-entropy of A is defined as

$$\mathsf{H}_\infty(A) = -\log(\max_a \Pr[A = a]).$$

And for two correlated random variables A, B, the conditional min-entropy of A given B is defined as

$$\tilde{\mathsf{H}}_\infty(A|B) = -\log\left(\mathbb{E}_{b \leftarrow B}\left[\max_a \Pr[A = a | B = b]\right]\right).$$

Definition 3.17 (Two-Source Extractor)**.** A deterministic function $\mathsf{Ext} : \{0,1\}^{n_1} \times \{0,1\}^{n_2} \to \{0,1\}^m$ is said to be a (k_1, k_2, ϵ)-2-source-extractor if, for every pair of independent random variables (X, Y) such that $\mathsf{H}_\infty(X) \geq k_1, \mathsf{H}_\infty(Y) \geq k_2$, the following condition holds:

$$\Delta(\mathsf{Ext}(X, Y), U_m) \leq \epsilon.$$

Moreover, Ext is said to be strong in the first input if

$$\Delta((X, \mathsf{Ext}(X, Y)), (X, U_m)) \leq \epsilon,$$

and strong in the second input if

$$\Delta((Y, \mathsf{Ext}(X, Y)), (Y, U_m)) \leq \epsilon.$$

Lemma 3.18 ([DORS08]). *If a random variable w has at most 2^k possible assignments, then $\tilde{\mathsf{H}}_\infty(X|w) \geq \mathsf{H}_\infty(X) - k$.*

Lemma 3.19 (Residual Entropy after Leakage [DORS08]). *Let $\ell = \ell(\kappa)$ be a leakage bound and X, Y be two independent random variables. For every 2-party BCL leakage $L \in \mathcal{L}^{2,0,\ell}_{\mathsf{BCL}}$ on a pair of independent random variables (X, Y), let $w \leftarrow L(X, Y)$ be the transcript of leakage execution L. Then for every $\epsilon > 0$, with probability at least $1 - 2\epsilon$ over the random choice of w, we have $\mathsf{H}_\infty(X|w) \geq \mathsf{H}_\infty(X) - \ell - \log(1/\epsilon)$, $\mathsf{H}_\infty(Y|w) \geq \mathsf{H}_\infty(Y) - \ell - \log(1/\epsilon)$ and $(X|w, Y|w)$ remain independent.*

3.5 Leakage-Resilient One-Time Pad

We give a quick recap of the notion of leakage-resilient one-time pad introduced by Goldwasser and Rothblum [GR12].

Definition 3.20 (Leakage-Resilient One-Time Pad). For message length $m = m(\kappa)$ and parameter $\lambda = \lambda(\kappa)$, a leakage-resilient one-time pad (LROTP for short) is a one-shot encryption scheme of length-m messages, consisting of the following functionalities:

- Gen. Outputs a random key $\mathsf{key} \xleftarrow{\$} \mathbb{F}_2^\lambda / \{0^\lambda\}$.
- $\mathsf{Enc}(\mathsf{key}, \boldsymbol{b})$. On secret key $\mathsf{key} \in \mathbb{F}_2^\lambda / \{0^\lambda\}$ and plaintext $\boldsymbol{b} \in \mathbb{F}_2^m$, encrypts $\boldsymbol{b}$ using the inner-product extractor. Concretely, outputs a uniformly random ciphertext $M \in \mathbb{F}_2^{\lambda \times m}$ conditioned on $\mathsf{key}^T M = \boldsymbol{b}$.
- $\mathsf{Dec}(\mathsf{key}, M)$. On secret key $\mathsf{key} \in \mathbb{F}_2^\lambda / \{0^\lambda\}$ and ciphertext $M \in \mathbb{F}_2^{\lambda \times m}$, decrypts $\boldsymbol{b} \leftarrow \mathsf{key}^T M$.
- oEnc. Outputs a uniformly random ciphertext $M \xleftarrow{\$} \mathbb{F}_2^{\lambda \times m}$.

We use $\mathsf{LROTP}_{\boldsymbol{b}}$ to denote the random variable consisting of a random secret key and a random ciphertext with underlying plaintext $\boldsymbol{b}$:

$$\mathsf{LROTP}_{\boldsymbol{b}} := \{(\mathsf{key}, M) : \mathsf{key} \leftarrow \mathsf{Gen}, M \leftarrow \mathsf{Enc}(\mathsf{key}, \boldsymbol{b})\}.$$

Similarly, we use $\mathsf{LROTP}_{\mathsf{rnd}}$ to denote the distribution of a random key-ciphertext pair:

$$\mathsf{LROTP}_{\mathsf{rnd}} := \{(\mathsf{key}, M) : \mathsf{key} \leftarrow \mathsf{Gen}, M \leftarrow \mathsf{oEnc}\}.$$

Lemma 5.4 of [GR12] states that LROTP is resilient to a BCL leakage attack on input sequence (key, M):

Lemma 3.21. *For all message lengths $m \leq 0.1\lambda$ and leakage bounds $\ell \leq 0.1\lambda$, for any pair of plaintexts $\boldsymbol{b}_1, \boldsymbol{b}_2 \in \{0,1\}^m$ and any BCL leakage function $L \in \mathcal{L}^{2,0,\ell}_{\mathsf{BCL}}$, the distribution of applying L to the input sequence $(\mathsf{key}, M) \xleftarrow{\$} \mathsf{LROTP}_{\boldsymbol{b}_1}$ is statistically close to that when $(\mathsf{key}, M) \xleftarrow{\$} \mathsf{LROTP}_{\boldsymbol{b}_2}$ with statistical distance $\exp(-\Omega(\lambda))$. By an averaging argument, the same also holds between $\mathsf{LROTP}_{\boldsymbol{b}}$ and $\mathsf{LROTP}_{\mathsf{rnd}}$ for any $\boldsymbol{b} \in \{0,1\}^m$.*

4 Semi-honest LTMPC in the $\mathcal{F}_{\mathsf{Mult}}$-Hybrid Model

In this section, we follow the approach of Section 2.1 to present a semi-honest n-LTMPC protocol π_{MLTMPC} in the $\mathcal{F}_{\mathsf{Mult}}$-hybrid model, where $\mathcal{F}_{\mathsf{Mult}}$ is a functionality for multiplication over additive sharings as defined below.

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

Running with parties $P_1, \ldots, P_n$ and a simulator $\mathcal{S}$, $\mathcal{F}_{\mathsf{Mult}}$ works as follows:

- Upon receiving input $(\mathsf{mul-input}, P_i, \langle\alpha\rangle_{n,i}, \langle\beta\rangle_{n,i})$ from every party P_i, notify $\mathcal{S}$ about the computation.
- Once $\mathcal{S}$ acknowledges, randomly sample $(\langle\gamma\rangle_{n,1}, \ldots, \langle\gamma\rangle_{n,n}) \in \mathbb{F}_2^n$ conditioned on

$$\sum_{i=1}^{n} \langle\gamma\rangle_{n,i} = \left(\sum_{i=1}^{n} \langle\alpha\rangle_{n,i}\right) \cdot \left(\sum_{i=1}^{n} \langle\beta\rangle_{n,i}\right)$$

  and send $(\mathsf{mul-output}, P_i, \langle\gamma\rangle_{n,i})$ to each P_i.

Figure 5: The Multiplication Functionality

4.1 Construction

At a high level, the protocol π_{MLTMPC} performs a standard semi-honest GMW protocol [Gol09, GMW87] to emulate a strong parity-LTC C_{pr}, and invokes oracle calls to $\mathcal{F}_{\mathsf{Mult}}$ to compute the multiplication gates.

We analyze the security of π_{MLTMPC} by considering different values of the corruption threshold t separately: we consider the case of $t \leq n-2$ in Section 4.2, and later in Section 4.3 we handle the case of exactly $t = n-1$ corruptions.

4.2 Security Proof of $t \leq n-2$

In this case, the strong leakage-tolerance property of C_{pr} ensures that the simulator $\mathcal{S}$ can locally sample an additive secret sharing of every internal wire. Building on the analysis of [GIM+16], we can establish the bounded communication leakage-tolerance property of π_{MLTMPC}. A more subtle issue arises when we deal with the input encoding of some $\boldsymbol{x}_i$ or output decoding of some $\boldsymbol{y}_i$. In this case, the corresponding party P_i effectively collects the whole secret sharing. As a result, the premise of Lemma 3.13 no longer holds. To this end, we make the following observation: a random additive secret sharing of input wire $\boldsymbol{x}_i$ (resp. output wire $\boldsymbol{y}_i$) can be obtained by sampling a random additive sharing $\langle 0\rangle_n$ and adding $\boldsymbol{x}_i$ (resp. $\boldsymbol{y}_i$) to P_i's share. Because the sharing $\langle 0\rangle_n$ does not include any private information about input $\boldsymbol{x}_i$ or output $\boldsymbol{y}_i$, it can be freely sampled by the simulator $\mathcal{S}$ before the protocol execution. The hybrid argument given below formalizes the above analysis:

Hybrid Real**.** This hybrid is the real-life execution.

Hybrid Hyb_1**.** In this hybrid, we first perform a real execution of C_{pr} on input $\boldsymbol{x} = (\boldsymbol{x}_i)_{i\in[n]}$ and a random auxiliary input $\boldsymbol{r}_{\mathcal{F}'} \xleftarrow{\$} \{0,1\}^{w\cdot|\boldsymbol{r}_{\mathcal{F}}|}$, and acquire a wire value assignment $W[C_{\mathsf{pr}}]$. For each wire w except the output wire of an addition gate, we sample a random additive secret sharing $\langle w\rangle_n$ of the corresponding wire-value variable in $W[C_{\mathsf{pr}}]$ as the share of each party. The secret sharing of output wires of addition gates can be computed accordingly.

The locally samplable property of the underlying BCLSS scheme and the perfect correctness of the ideal functionality $\mathcal{F}_{\mathsf{Mult}}$ ensure that Real and Hyb_1 are identical.

Protocol π_{MLTMPC}

Parties $P_1, \ldots, P_n$ evaluate an n-party randomized function $\mathcal{F}$:

- Construct a strong parity-LTC C_{pr} of $\mathcal{F}'$, a *deterministic* function modified from $\mathcal{F}$ introduced as follows:
  1. Let $(\mathsf{Enc}_{\mathsf{pr}}, \mathsf{Dec}_{\mathsf{pr}})$ be the inner-product encoding scheme as defined in Corollary 3.15, such that the additive secret sharing of $(\mathsf{Enc}_{\mathsf{pr}}, \mathsf{Dec}_{\mathsf{pr}})$ induces a perfectly simulatable and locally samplable $(n, \ell, \epsilon/2)$-BCLSS scheme with encoding length $w = O(\ell + \log(1/\epsilon))$ per bit for some $\epsilon = \mathsf{negl}(\kappa)$. The functionality $\mathcal{F}'$ on input $\boldsymbol{x}$ and an auxiliary input $\boldsymbol{r}_{\mathcal{F}'} \in \{0,1\}^{w \cdot |\boldsymbol{r}_{\mathcal{F}}|}$ computes $\mathcal{F}'(\boldsymbol{x}, \boldsymbol{r}_{\mathcal{F}'}) := \mathcal{F}(\boldsymbol{x}; \mathsf{Dec}_{\mathsf{pr}}(\boldsymbol{r}_{\mathcal{F}'}))$, where $|\boldsymbol{r}_{\mathcal{F}}|$ is the length of internal randomness of $\mathcal{F}$.
  2. Let C_{pr} be a strong parity-LTC of $\mathcal{F}'$. Append a "trivial" multiplication gate $y' \leftarrow y \cdot y$ to each output wire y in C_{pr}, and use y' as the real output.
- $P_1, \ldots, P_n$ compute an additive secret sharing of each wire of C_{pr} in topological order as follows:
  1. For each input wire $\boldsymbol{x}_i$ carrying the input of party P_i, P_i randomly samples an additive sharing $\langle \boldsymbol{x}_i \rangle_n$ and distributes it to other parties. For the auxiliary input $\boldsymbol{r}_{\mathcal{F}'}$, each party simply samples a random string of length $w \cdot |\boldsymbol{r}_{\mathcal{F}}|$ as its share $\langle \boldsymbol{r}_{\mathcal{F}'} \rangle_{n,i}$.
  2. For each random wire r of C_{pr}, each party P_i locally samples a bit as its share $\langle r \rangle_{n,i}$.
  3. For each addition gate $g \in C_{\mathsf{pr}}$ with input wires α, β and output wire γ, each party locally adds its shares of $\langle \alpha \rangle_n$ and $\langle \beta \rangle_n$ to obtain $\langle \gamma \rangle_n$.
  4. For each multiplication gate $g \in C_{\mathsf{pr}}$ with input wires α, β and output wire γ, each party P_i sends its shares $\langle \alpha \rangle_{n,i}$ and $\langle \beta \rangle_{n,i}$ to $\mathcal{F}_{\mathsf{Mult}}$ and receives $\langle \gamma \rangle_{n,i}$.
  5. For each output wire $\boldsymbol{y}_i$ carrying the output of party P_i, each party $P_j \neq P_i$ delivers its share $\langle \boldsymbol{y}_i \rangle_{n,j}$ to P_i so that P_i can reconstruct $\boldsymbol{y}_i$.

Figure 6: n-LTMPC in $\mathcal{F}_{\mathsf{Mult}}$-Hybrid Model

Hybrid Hyb_2. In this hybrid, we invoke the simulator $\mathcal{S}_{\mathsf{pr}}$ to obtain a state translation function $T_{\mathsf{pr}}(\boldsymbol{x}, \boldsymbol{r}_{\mathcal{F}'}, \boldsymbol{y} \leftarrow \mathcal{F}'(\boldsymbol{x}, \boldsymbol{r}_{\mathcal{F}'})) \in \mathsf{NC}^0[1]$ (W.L.O.G. we may assume that the component T^w_{pr} of an input wire or output wire w always outputs the wire-value variable itself). Then we replace the additive secret sharing $\langle w \rangle_n$ used in Hyb_1 with a random secret sharing $\langle T^w_{\mathsf{pr}}(\boldsymbol{x}, \boldsymbol{r}_{\mathcal{F}'}, \boldsymbol{y}) \rangle_n$. Note that in Hyb_2, the sharing of each output wire $\langle \boldsymbol{y}_i \rangle_n$ is directly computed from the ideal functionality output because we let $\boldsymbol{y}_i$ undergo a trivial multiplication gate.

The strong leakage-tolerance property of C_{pr} promises that even conditioned on $(\boldsymbol{x}, \boldsymbol{r}_{\mathcal{F}'}, \boldsymbol{y})$, any parity leakage adversary can distinguish $(W[C_{\mathsf{pr}}], T_{\mathsf{pr}}(\boldsymbol{x}, \boldsymbol{r}_{\mathcal{F}'}, \boldsymbol{y}))$ with no more than $2^{-\ell/2-1}\epsilon$ advantage. Hence, we can apply Lemma 3.13 to the wire values besides input wires and output wires to show that Hyb_1 and Hyb_2 are $\epsilon/2$-indistinguishable.

Hybrid Hyb_3. In this hybrid, we set $\boldsymbol{r}_{\mathcal{F}'}$ to be a random parity encoding of the all-0 string $\mathsf{Enc}_{\mathsf{pr}}(0^{|\boldsymbol{r}_{\mathcal{F}}|})$ prepared in advance, and compute the output directly from the ideal functionality $\boldsymbol{y} \leftarrow \mathcal{F}(\boldsymbol{x})$. In this way, we can view T_{pr} as a function of $(\boldsymbol{x}, \boldsymbol{y})$.

Note that in Hyb_2 each party P_i is only provided with a share of the n-BCLSS encryption of $\boldsymbol{r}_{\mathcal{F}}$. The security of the underlying BCLSS scheme ensures that Hyb_2 and Hyb_3 are $\epsilon/2$-indistinguishable.

Hybrid Ideal. This hybrid is the ideal world execution. In this hybrid, we construct a simulator $\mathcal{S}$ to emulate the secret sharing generation process of Hyb_3 in the following way:

- Invoke the simulator $\mathcal{S}_{\mathsf{pr}}$ to obtain T_{pr}. Recall that T_{pr} is an $\mathsf{NC}^0[1]$ function, every T^w_{pr} is of the form $u^w a^w + v^w$, where $u^w, v^w \in \mathbb{F}_2$ are constants and $a^w \in \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \cup \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n\}$.
- For each wire w except the output wires of addition gates, randomly sample an additive secret sharing $\langle v^w \rangle_n$.
- Output n state translation functions $T_1, \ldots, T_n$, such that for each wire w which is not an output wire of an addition gate, T_i outputs $u^w a^w + \langle v^w \rangle_{n,i}$ as its share if $a^w \in \{\boldsymbol{x}_i, \boldsymbol{y}_i\}$, or simply outputs $\langle v^w \rangle_{n,i}$ otherwise. If w is an input wire or output wire, the corresponding party P_i also hardcodes the shares $\langle v^w \rangle_{n,[n]/\{i\}}$ as the messages sent to or received from other parties.

4.3 Security proof of $t = n - 1$

When exactly $n - 1$ parties are corrupted, the BCLSS scheme no longer provides privacy against leakage queries. In this case, we exploit the adaptive security of the semi-honest GMW protocol [CLOS02] to present a simulator $\mathcal{S}$ that *faithfully* emulates a real execution:

1. $\mathcal{S}$ submits the inputs of all corrupted parties (say, they are $P_1, \ldots, P_{n-1}$) to the ideal functionality $\mathcal{F}$ and receives the ideal functionality output of the corrupted parties. Then $\mathcal{S}$ emulates the views of all corrupted parties using the following strategy: for each input wire $\boldsymbol{x}_i$ (resp. output wire $\boldsymbol{y}_i$) such that $i \in \{1, \ldots, n-1\}$, it simply samples a random $\langle \boldsymbol{x}_i \rangle_n$ (resp. $\langle \boldsymbol{y}_i \rangle_n$) as the secret sharing; and for each other wire w (except for the output wire of an addition gate), it uses $n - 1$ random bits as the secret sharing of the corrupted parties.
2. $\mathcal{S}$ constructs a state translation function T_n of the honest party P_n. Remember that T_n can access the entire input $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ and the entire output $\boldsymbol{y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)$; if the functionality $\mathcal{F}$ is randomized, it also gains valid internal randomness $\boldsymbol{r}_{\mathcal{F}}$ such that $\boldsymbol{y} = \mathcal{F}(\boldsymbol{x}; \boldsymbol{r}_{\mathcal{F}})$ as $\mathcal{F}$ is adaptively well-formed. T_n then emulates the view of P_n as follows:

- For each input wire or output wire w of functionality $\mathcal{F}$, if it carries an input or output bit of a corrupted party, the share $\langle w \rangle_{n,n}$ is determined in Step 1; otherwise, set the share of P_n as $w + \sum_{i=1}^{n-1} \langle w \rangle_{n,i}$.
- For the input wire corresponding to the auxiliary input $\boldsymbol{r}_{\mathcal{F}'}$, generate the share $\langle \boldsymbol{r}_{\mathcal{F}'} \rangle_{n,n}$ using $\mathsf{SimSS}(n, \langle \boldsymbol{r}_{\mathcal{F}'} \rangle_{n,[n]/n}, \boldsymbol{r}_{\mathcal{F}})$. This step crucially relies on the perfectly simulatable property of the underlying BCLSS scheme.
- Preform an honest execution on input $\boldsymbol{x}$ and auxiliary input $\boldsymbol{r}_{\mathcal{F}'}$ of C_{pr} to obtain an assignment $W[C_{\mathsf{pr}}]$. With overwhelming probability, the output of C_{pr} equals $\boldsymbol{y}$.
- For each other wire w (except for the output wire of an addition gate), simply fix each share $\langle w \rangle_{n,n}$ that has not yet been determined in Step 1 as $w + \sum_{i=1}^{n-1} \langle w \rangle_{n,i}$, where w is the corresponding assignment in $W[C_{\mathsf{pr}}]$ and $\langle w \rangle_{n,i}$ ($i \in \{1, \ldots, n-1\}$) are the shares sampled by $\mathcal{S}$ in Step 1.

Recall that [MS26] constructs strong parity-LTCs C_{pr} with exponentially small security loss. We conclude the above discussion with the following theorem:

Theorem 4.1. *Let $\ell = \ell(\kappa)$ be a leakage bound. Then, for any number of parties $n \geq 2$ and any adaptively well-formed n-party functionality $\mathcal{F}$, the protocol π_{MLTMPC} is a semi-honest $(n, n-1, \ell)$-LTMPC protocol of $\mathcal{F}$ in the $\mathcal{F}_{\mathsf{Mult}}$-hybrid model.*

4.4 Application: Bounded Communication Leakage-Tolerant Circuits

An important application of Theorem 4.1 is to construct a bounded communication leakage-tolerant circuit (or BCL-LTC for short). Briefly speaking, an ℓ-BCL-LTC C_{BCL} is a circuit in which there is a partition of its wires into two disjoint parts such that any $\mathcal{L}_{\mathsf{BCL}}^{2,0,\ell}$-leakage attack applied to the 2 partitions can be emulated by another $\mathcal{L}_{\mathsf{BCL}}^{2,0,\ell}$-leakage attack on the 2 partitions of the ideal input and output $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2), \boldsymbol{y} = (\boldsymbol{y}_1, \boldsymbol{y}_2)$. To align with the notion of LTMPC, we additionally require the BCL-LTC to also be leakage-obliviously simulatable. Namely, there exists a simulator $\mathcal{S}$ which outputs 2 state translation functions T_1, T_2, where each T_i takes the ideal state $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ as input and outputs a simulated view of the i-th partition, such that the simulated views can fool any $\mathcal{L}_{\mathsf{BCL}}^{2,0,\ell}$-leakage adversary.

Definition 4.2 (Bounded Communication Leakage-Tolerant Circuits)**.** Let C be a circuit implementation of a (possibly randomized) function $\mathcal{F}$ defined as $\boldsymbol{y} \leftarrow \mathcal{F}(\boldsymbol{x})$. Let $(\boldsymbol{x}, \boldsymbol{y}) = ((\boldsymbol{x}_1, \boldsymbol{y}_1), (\boldsymbol{x}_2, \boldsymbol{y}_2))$ be a 2-partition of the input and output. Given a leakage bound $\ell = \ell(\kappa)$, we say that C is an ℓ-BCL-LTC, if there exists a 2-partition $W = (W_1, W_2)$ of the circuit wires in C and a simulator $\mathcal{S}$, such that for every input assignment to $\boldsymbol{x}$ and every adversary $\mathcal{A}$, the outputs of the following two experiments are statistically indistinguishable:

$$\mathsf{Real}_{\mathcal{A},\mathcal{L},\boldsymbol{x}} \approx_S \mathsf{Ideal}_{\mathcal{S},\mathcal{A},\mathcal{L},\boldsymbol{x}}.$$

We abbreviate the notation to BCL-LTC if ℓ is clear in the context.

Construction. Viewing $\mathcal{F}$ as a 2-party function, we can interpret the semi-honest $(2, 0, \ell)$-LTMPC protocol of $\mathcal{F}$ in the $\mathcal{F}_{\mathsf{Mult}}$-hybrid model as an ℓ-BCL-LTC. Specifically, let π_{MLTMPC} be the LTMPC protocol over parties $\{P_1, P_2\}$. We assign each additive share of P_1 to W_1 and assign each share of P_2 to W_2. The only remaining problem is how to handle the multiplication functionality $\mathcal{F}_{\mathsf{Mult}}$. As observed in [GIW17, IS25], the multiplication gate over additive sharings

Experiment $\mathsf{Real}_{\mathcal{A},\mathcal{L},\boldsymbol{x}}$

- Run C on input $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2)$ and obtain $\boldsymbol{y} = (\boldsymbol{y}_1, \boldsymbol{y}_2), W = (W_1, W_2)$.
- $(\boldsymbol{y}_1, \boldsymbol{y}_2) \to \mathcal{A}$.
- $\mathcal{A}$ outputs a leakage function $L \in \mathcal{L}_{\mathsf{BCL}}^{2,0,\ell}$.
- $L(W_1, W_2) \to \mathcal{A}$.
- Output $(\mathsf{view}_{\mathcal{A}}, \boldsymbol{y})$.

Experiment $\mathsf{Ideal}_{\mathcal{S},\mathcal{A},\mathcal{L},\boldsymbol{x}}$

- Compute $(\boldsymbol{y}_1, \boldsymbol{y}_2) \leftarrow \mathcal{F}(\boldsymbol{x}_1, \boldsymbol{x}_2)$.
- $\mathcal{S}$ outputs state translation functions T_1, T_2.
- $(\boldsymbol{y}_1, \boldsymbol{y}_2) \to \mathcal{A}$.
- $\mathcal{A}$ outputs a leakage function $L \in \mathcal{L}_{\mathsf{BCL}}^{2,0,\ell}$.
- $L(T_1(\boldsymbol{x}_1, \boldsymbol{y}_1), T_2(\boldsymbol{x}_2, \boldsymbol{y}_2)) \to \mathcal{A}$.
- Output $(\mathsf{view}_{\mathcal{A}}, \boldsymbol{y})$.

Figure 7: Bounded Communication Leakage-Tolerant Circuit Experiments

(say, $(\gamma_1, \gamma_2) \leftarrow g((\alpha_1, \beta_1), (\alpha_2, \beta_2))$) can be implemented by a sub-circuit in which every wire can be perfectly simulated by either $(\alpha_1, \beta_1, \gamma_1)$ or $(\alpha_2, \beta_2, \gamma_2)$ (corresponding to either W_1 or W_2). Substituting this information-theoretically secure sub-circuit into π_{MLTMPC}, we obtain the desired BCL-LTC C_{BCL}.

5 Computationally Secure Semi-honest LTMPC

This section discusses the construction of computationally secure semi-honest LTMPCs in an ideal leakage-tolerant oblivious transfer (LTOT for short) hybrid model $\mathcal{F}_{\mathsf{OT}}$. We present an outline of the contents: in Section 5.1 we realize $\mathcal{F}_{\mathsf{Mult}}$ in the two-party setting by merely restating a theorem in [GIM+16]; building upon the 2-LTMPC result, in Section 5.2 we provide an efficient implementation of $\mathcal{F}_{\mathsf{Mult}}$ for any constant number of parties n; finally, in Section 5.3 we construct a compiler which reduces generic n-party LTMPC to constant party LTMPC (to be more precise, our compiler makes black-box use of an m-LTMPC protocol with $m \leq 4$).

5.1 2-Party LTMPC

We first present the protocol π^2_{Mult} instantiating a 2-party $\mathcal{F}_{\mathsf{Mult}}$ functionality. The protocol is a restatement of the result in [GIM+16].

Security. For completeness, we provide a simulation-based security proof of π^2_{Mult}. The simulator $\mathcal{S}$ honestly emulates each P_i's view using only its ideal state $(\alpha_i, \beta_i, \gamma_i)$. For more details:

- To emulate P_1's view, $\mathcal{S}$ honestly computes $\gamma(u, v) = (\alpha_1 + u) \cdot (\beta_1 + v) + \gamma_1$ for all $(u, v) \in \mathbb{F}_2^2$ and takes $(0, \gamma(u, v))$ as the input of the (u, v)-th $\mathcal{F}_{\mathsf{OT}}$ call.
- To emulate P_2's view, observe that the output bit of each $\mathcal{F}_{\mathsf{OT}}$ call can be rewritten as $b_{u,v} \cdot \gamma_2$ and can be computed via $(\alpha_2, \beta_2, \gamma_2)$.

Combining this with Theorem 4.1, we immediately obtain a 2-party LTMPC protocol in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model:

Lemma 5.1. *Let $\ell = \ell(\kappa)$ be a leakage bound. Then every adaptively well-formed 2-party functionality $\mathcal{F}$ admits an unconditionally secure semi-honest $(2, 1, \ell)$-LTMPC implementation in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model.*

Circuit C_{BCL}

1. **Input:** The circuit C_{BCL} takes $(\boldsymbol{x}_1, \boldsymbol{x}_2)$ as a 2-partition input.
2. **Evaluation:** Let π_{MLTMPC} be a $(2, 0, \ell)$-LTMPC protocol in the $\mathcal{F}_{\mathsf{Mult}}$-hybrid model (Figure 6) realizing $(\boldsymbol{y}_1, \boldsymbol{y}_2) \leftarrow \mathcal{F}(\boldsymbol{x}_1, \boldsymbol{x}_2)$. C_{BCL} then emulates the computation of π_{MLTMPC} as follows:
   - For each input wire x carrying bits of $\boldsymbol{x}_i$, C_{BCL} samples a random bit as $\langle x \rangle_{2,3-i}$ and computes $\langle x \rangle_{2,i} = x - \langle x \rangle_{2,3-i}$. For the auxiliary input $\boldsymbol{r}_{\mathcal{F}'}$, C_{BCL} samples two random strings as $\langle \boldsymbol{r}_{\mathcal{F}'} \rangle_2$.
   - For each additive sharing of random wire $\langle r \rangle_2$, C_{BCL} generates two random bits as $\langle r \rangle_{2,1}, \langle r \rangle_{2,2}$ respectively.
   - For each addition gate g with input sharings $\langle \alpha \rangle_2, \langle \beta \rangle_2$ and output sharing $\langle \gamma \rangle_2$, C_{BCL} locally adds the shares to obtain $\langle \gamma \rangle_2$.
   - For each multiplication gate g with input sharings $\langle \alpha \rangle_2, \langle \beta \rangle_2$ and output sharing $\langle \gamma \rangle_2$, C_{BCL} sets $\langle \gamma \rangle_{2,1}$ to be a random bit. Then for all $(u, v) \in \mathbb{F}_2^2$, C_{BCL} computes
   $$\begin{aligned} a_{u,v} &= (\langle \alpha \rangle_{2,1} + u) \cdot (\langle \beta \rangle_{2,1} + v) + \langle \gamma \rangle_{2,1} \\ b_{u,v} &= (\langle \alpha \rangle_{2,2} + u + 1) \cdot (\langle \beta \rangle_{2,2} + v + 1) \\ c_{u,v} &= a_{u,v} \cdot b_{u,v} \end{aligned}$$
   Then C_{BCL} computes $\langle \gamma \rangle_{2,2} = \sum_{(u,v) \in \mathbb{F}_2^2} c_{u,v}$.
   - For each output wire y carrying bits of $\boldsymbol{y}_i$, C_{BCL} accumulates $\langle y \rangle_{2,1}$ and $\langle y \rangle_{2,2}$ to recover y.
3. **Output:** C_{BCL} simply outputs $(\boldsymbol{y}_1, \boldsymbol{y}_2)$.

Figure 8: Bounded Communication Leakage-Tolerant Circuits

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

Running with parties S, R and a simulator $\mathcal{S}$, $\mathcal{F}_{\mathsf{OT}}$ works as follows:

- Upon receiving input $(\mathsf{send}, S, R, x_0, x_1)$ where $x_0, x_1 \in \{0, 1\}$ from party S, send (send, S, R) to $\mathcal{S}$ and record (x_0, x_1).
- Upon receiving input $(\mathsf{receive}, S, R, b)$ where $b \in \{0, 1\}$ from party R, send $(\mathsf{receive}, S, R)$ to $\mathcal{S}$.
- Once $\mathcal{S}$ acknowledges, output the value x_b to R.

Figure 9: The OT Functionality

Protocol π^2_{Mult}

On input $(\alpha_i := \langle\alpha\rangle_{2,i}, \beta_i := \langle\beta\rangle_{2,i})$ $(i \in \{1,2\})$ of each party, P_1, P_2 evaluate the 2-party function $\mathcal{F}_{\mathsf{Mult}}$:

- P_1 randomly samples $\gamma_1 \xleftarrow{\$} \mathbb{F}_2$ and computes

$$\gamma(u,v) := (\alpha_1 + u) \cdot (\beta_1 + v) + \gamma_1$$

  for every $(u,v) \in \mathbb{F}_2^2$.
- P_1 and P_2 run 4 invocations of $\mathcal{F}_{\mathsf{OT}}$ (labeled by $(u,v) \in \mathbb{F}_2^2$), where P_1 acts as the sender and P_2 acts as the receiver. In the invocation (u,v), P_1 prepares its input $(0, \gamma(u,v))$, while P_2 sends input $b_{u,v} := [u = \alpha_2 \wedge v = \beta_2]$ to $\mathcal{F}_{\mathsf{OT}}$ and receives an output bit.
- P_1 outputs γ_1 and P_2 outputs the bit received in invocation (α_2, β_2) as γ_2.

Figure 10: Semi-Honest 2-Party Multiplication Protocol

5.2 Constant-Party LTMPC

Next, for any number of parties $n \geq 3$, we provide a protocol π^n_{Mult} that securely realizes the n-party $\mathcal{F}_{\mathsf{Mult}}$ functionality, assuming a secure $(n-1)$-LTMPC protocol, as well as a 2-LTMPC protocol for arbitrary functionalities. We present the formal protocol below:

Complexity. The protocol π^n_{Mult} recursively invokes two $(n-1)$-LTMPC protocols and one 2-LTMPC protocol, each computing a functionality of size $\mathsf{poly}(n, \kappa, \ell, \log(1/\epsilon))$. Therefore, the running time of π^n_{Mult} is bounded by some polynomial $\mathsf{poly}(n, \kappa, \ell, \log(1/\epsilon))$ for every constant n, assuming an efficient 2-LTMPC protocol.

Security. For every $n \geq 3$, we prove the security of protocol π^n_{Mult} in the $((n-1)$-LTMPC, 2-LTMPC)-hybrid model by constructing a simulator $\mathcal{S}$ together with n state translation functions $T_1, \ldots, T_n$. To achieve this, let S_L, S_R denote the numbers of corrupted parties among $P_1, \ldots, P_{n-1}$ and among $P_1, \ldots, P_{n-2}, P_n$, respectively. We treat the cases according to the values of S_L and S_R:

1. $S_L \geq n-2$**:** In this case, $P_1, \ldots, P_{n-1}$ can honestly compute the shares of every $\gamma(u,v)$ in accordance with the perfectly simulatable property of the underlying BCLSS scheme: if every party is corrupted, $\mathcal{S}$ can directly obtain the share $\langle\gamma(u,v)\rangle^{\mathsf{BCL}}_{n-1}$ through direct computation; otherwise, suppose that P_i is the unique honest party. Given the ideal states $(\alpha_\cdot, \beta_\cdot, \gamma_\cdot)$ of corrupted parties, $\mathcal{S}$ first samples random strings as the shares $\gamma(u,v)_{[n-1]/i}$ of parties $P_1, \ldots, P_{i-1}, P_{i+1}, \ldots, P_{n-1}$ for each $(u,v) \in \mathbb{F}_2^2$. Then, T_i computes the share $\gamma(u,v)_i \xleftarrow{\$} \mathsf{SimSS}(i, \gamma(u,v)_{[n-1]/i}, \gamma(u,v))$ after locally computing $\gamma(u,v)$.

   A remaining tough issue is to let P_n learn the $(n-1)$-th share $\gamma(\alpha_n, \beta_n)_{n-1}$ in the case where P_{n-1} is the unique honest party among $P_1, \ldots, P_{n-1}$. In our simulation process, the random variable indicating $\gamma(u,v)_{n-1}$ depends only on a private bit $\gamma(u,v) \in \mathbb{F}_2$. As a result, we can let $\mathcal{S}$ generate a tableau of random assignments $(w_{u,v,0} \in \langle 0\rangle^{\mathsf{BCL}}_{n-1}, w_{u,v,1} \in \langle 1\rangle^{\mathsf{BCL}}_{n-1})_{(u,v)\in\mathbb{F}_2^2}$ s.t. $w_{u,v,0}$ and $w_{u,v,1}$ enjoy the same corrupted parties' shares, without knowing the correct output bit $\gamma(u,v)$. This enables each corrupted party P_i $(i \in [n-2])$ to simply use the counterpart

Protocol π^n_{Mult}

On input $(\alpha_i := \langle\alpha\rangle_{n,i}, \beta_i := \langle\beta\rangle_{n,i})$ $(i \in [n])$ of each party, $P_1, \ldots, P_n$ evaluate the n-party function $\mathcal{F}_{\mathsf{Mult}}$ for some constant n:

- Each of the parties $P_1, P_2, \ldots, P_{n-1}$ locally samples its output $\gamma_i \xleftarrow{\$} \mathbb{F}_2$ $(i \in [n-1])$. Then they perform a secure $(n-1)$-LTMPC protocol to compute a random $(n-1)$-BCLSS $\langle\gamma(u,v)\rangle^{\mathsf{BCL}}_{n-1} := (\gamma(u,v)_1, \ldots, \gamma(u,v)_{n-1})$ of

$$\gamma(u,v) := \left(u + \sum_{i=1}^{n-1} \alpha_i\right) \cdot \left(v + \sum_{i=1}^{n-1} \beta_i\right) + \sum_{i=1}^{n-1} \gamma_i$$

  for every $(u,v) \in \mathbb{F}_2^2$.
- P_n runs a 2-LTMPC protocol with P_{n-1} to learn $\gamma(\alpha_n, \beta_n)_{n-1}$.
- $P_1, \ldots, P_{n-2}$ and P_n run an $(n-1)$-LTMPC protocol computing the decoding functionality $\mathcal{F}_{\mathsf{Dec}}$ so that P_n is able to decode the output $\gamma_n = \gamma(\alpha_n, \beta_n)$.
- Each party P_i outputs γ_i.

Figure 11: Semi-Honest Constant-Party Multiplication Protocol

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

Running with parties $P_1, \ldots, P_{n-2}, P_n$ and a simulator $\mathcal{S}$, $\mathcal{F}_{\mathsf{Dec}}$ works as follows:

- Upon receiving input $\left(\mathsf{share}, P_i, \{\gamma(u,v)_i\}_{(u,v)\in\mathbb{F}_2^2}\right)$ from each of the parties $P_1, \ldots, P_{n-2}$ and input $(\mathsf{share}, P_n, \alpha, \beta, \gamma(\alpha,\beta)_{n-1})$ from P_n, notify to $\mathcal{S}$ about the computation.
- Once $\mathcal{S}$ acknowledges, parse $(\gamma(\alpha,\beta)_1, \ldots, \gamma(\alpha,\beta)_{n-1})$ as an $(n-1)$-BCLSS $\langle\gamma\rangle^{\mathsf{BCL}}_{n-1}$ and perform the decoding procedure $\mathsf{Dec}_{\mathsf{BCL}}$ to reconstruct the output γ.
- Send $(\mathsf{recon}, P_n, \gamma)$ to party P_n.

Figure 12: The Decoding Functionality

$w_{u,v,0}$ as its share $\gamma(u,v)_i$, and enables P_{n-1} to use the counterpart of $w_{u,v,\gamma(u,v)}$ as its share $\gamma(u,v)_{n-1}$. Recall that P_n receives the ideal functionality output $\gamma_n = \gamma(\alpha_n, \beta_n)$; it can also learn $\gamma(\alpha_n, \beta_n)_{n-1}$ by checking $w_{\alpha_n,\beta_n,\gamma_n}$ of the tableau.

2. $S_L < n-2$ **and** $S_R < n-2$**:** In this case, the security of the underlying $(n-1)$-BCLSS scheme ensures that for any bit $b \in \{0,1\}$, there exists no ℓ-leakage adversary that can distinguish a real share $\langle b \rangle_{n-1}^{\mathsf{BCL}}$ from a fake share $\langle 0 \rangle_{n-1}^{\mathsf{BCL}}$ with non-negligible advantage. Hence, the simulator $\mathcal{S}$ may replace the real share $\langle \gamma(u,v) \rangle_{n-1}^{\mathsf{BCL}}$ with $\langle 0 \rangle_{n-1}^{\mathsf{BCL}}$, while the decoding functionality $\mathcal{F}_{\mathsf{Dec}}$ is hardwired to output γ_n for P_n.

3. $S_L < n-2$ **and** $S_R \geq n-2$**:** This case is analogous to Case 2, except that $\langle \gamma(\alpha_n, \beta_n) \rangle_{n-1}^{\mathsf{BCL}}$ must be honestly generated. Note that the only possibility for Case 3 is that party P_n is corrupted while P_{n-1} is honest, implying that the ideal state $(\alpha_n, \beta_n, \gamma_n)$ becomes part of the input for $\mathcal{S}$. Thus, the simulator $\mathcal{S}$ can generate fake shares for every $(u,v) \neq (\alpha_n, \beta_n)$ as in Case 2, but sample the real share $\langle \gamma(\alpha_n, \beta_n) \rangle_{n-1}^{\mathsf{BCL}} = \langle \gamma_n \rangle_{n-1}^{\mathsf{BCL}}$ for $(u,v) = (\alpha_n, \beta_n)$.

Combining this with Theorem 4.1, we conclude with the following lemma:

Lemma 5.2. *Let $\ell = \ell(\kappa)$ be a leakage bound. Then for any constant $n \geq 2$, every adaptively well-formed n-party functionality $\mathcal{F}$ admits an unconditionally secure semi-honest $(n, n-1, \ell)$-LTMPC implementation in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model.*

5.3 Generic Solution

Finally, we use the technique of party virtualization to construct a generic n-party LTMPC protocol π_{LTMPC} given oracle calls to 4-LTMPC protocols. The protocol π_{LTMPC} emulates an execution of a semi-honest GMW protocol π_{Ext} over $m = n(n-1)/2$ virtual parties $V_{i,j}$, where the virtual party $V_{i,j}$ consists of two actual parties P_i and P_j.

Security. We show that our protocol π_{LTMPC} privately computes the n-party functionality $\mathcal{F}$ in a leakage-tolerant way by discussing the number of corrupted parties. Let $\mathcal{H}_{4,\ell}$ be a hybrid model containing the secure m-LTMPC protocols under ℓ leakage for $m \in \{2,3,4\}$. We have the following two lemmas:

Lemma 5.3. *For every adaptively well-formed n-party functionality $\mathcal{F}$, the protocol π_{LTMPC} is an unconditionally secure semi-honest n-LTMPC implementation of $\mathcal{F}$ against exactly $t = n-1$ corruptions and ℓ-bit leakage in the $H_{4,\ell}$-hybrid model.*

Suppose that P_i is the unique honest party and let P_j be any corrupted party. We view $V_{i,j}$ as an honest virtual party whose internal state is under leakage attack and mark all other virtual parties as corrupted. Then the proof of Lemma 5.3 is substantially identical to the analysis of LTMPC protocols against $t = n-1$ corruptions in the $\mathcal{F}_{\mathsf{Mult}}$-hybrid model (Theorem 4.1), and we omit it here.

Lemma 5.4. *For every n-party functionality $\mathcal{F}$, the protocol π_{LTMPC} is an unconditionally secure semi-honest $(n, n-2, \ell)$-LTMPC protocol computing $\mathcal{F}$ in the $\mathcal{H}_{4,\ell}$-hybrid model.*

Proof. We use a sequence of hybrids to prove Lemma 5.4.

Hybrid Real**.** This hybrid is the real-life execution.

Protocol π_{LTMPC}

$P_1, \ldots, P_n$ evaluate an n-party function $\mathcal{F}$:

- Construct a semi-honest GMW protocol π_{Ext} over $m = n(n-1)/2$ virtual parties $V_{i,j} = V_{j,i}$ $(1 \le i < j \le n)$ in the OT-hybrid model, which computes the randomized functionality $\mathcal{F}'$ defined as follows:

  1. On input $\boldsymbol{x}_{i,j} \parallel \boldsymbol{x}_{j,i}$ of each virtual party $V_{i,j}$, $\mathcal{F}'$ extracts the input $\boldsymbol{x}_i$ of each party P_i:
  $$\boldsymbol{x}_i = \sum_{j \neq i} \boldsymbol{x}_{i,j}.$$
  2. $\mathcal{F}'$ invokes the functionality $\mathcal{F}$ to compute the output $\boldsymbol{y}_i$ and samples a random $(n-1)$-additive secret sharing $\langle \boldsymbol{y}_i \rangle_{n-1} = (\boldsymbol{y}_{i,j})_{j \neq i}$.
  3. Each $V_{i,j}$ obtains output $\boldsymbol{y}_{i,j} \parallel \boldsymbol{y}_{j,i}$.

- Each party P_i prepares a random additive secret sharing of input $\boldsymbol{x}_i$ as $(\boldsymbol{x}_{i,j})_{j \neq i}$ used in π_{Ext}.
- Let $(\mathsf{Enc}_{\mathsf{BCL}}, \mathsf{Dec}_{\mathsf{BCL}})$ be a perfectly simulatable and locally samplable $(2, \ell)$-BCLSS scheme. For each virtual party $V_{i,j}$, P_i and P_j independently sample shares $\boldsymbol{r}^i_{i,j}$ and $\boldsymbol{r}^j_{i,j}$ as a 2-BCLSS $\langle \boldsymbol{r}_{i,j} \rangle^{\mathsf{BCL}}_2 = (\boldsymbol{r}^i_{i,j}, \boldsymbol{r}^j_{i,j})$ of $\boldsymbol{r}_{i,j}$, the internal randomness of $V_{i,j}$.
- $P_1, \ldots, P_n$ emulate the protocol π_{Ext}, where the transcript of each $V_{i,j}$ is split and stored as a 2-BCLSS $\langle \tau_{i,j} \rangle^{\mathsf{BCL}}_2 = (\tau^i_{i,j}, \tau^j_{i,j})$ between P_i and P_j. Specifically, each instruction of π_{Ext} is either: (1) instruct sender V_{i_1,j_1} to send a message to receiver V_{i_2,j_2}; (2) instruct sender V_{i_1,j_1} and receiver V_{i_2,j_2} to perform an OT call; (3) $V_{i,j}$ outputs $\boldsymbol{y}_{i,j} \parallel \boldsymbol{y}_{j,i}$. Since sending a message can be viewed as a special case of an OT call, we only present the implementation of the last two cases:

  1. For an OT call (x_0, x_1, b) between sender V_{i_1,j_1} and receiver V_{i_2,j_2}, $\{P_{i_1}, P_{j_1}, P_{i_2}, P_{j_2}\}$ run a 4-LTMPC to compute the following functionality $\mathcal{F}_{\mathsf{Int}}$ [a]. In this way, P_{i_2} and P_{j_2} can update $\langle \tau_{i_2,j_2} \rangle^{\mathsf{BCL}}_2$.
  2. For an output instruction of $V_{i,j}$, since the output $\boldsymbol{y}_{i,j} \parallel \boldsymbol{y}_{j,i}$ is just a deterministic function of $\boldsymbol{x}_{i,j} \parallel \boldsymbol{x}_{j,i}, \boldsymbol{r}_{i,j}$ and the transcript of $V_{i,j}$, P_i and P_j run 2-LTMPC to call $\mathcal{F}_{\mathsf{Int}}$. $\mathcal{F}_{\mathsf{Int}}$ decodes $\boldsymbol{y}_{i,j}$ for P_i and decodes $\boldsymbol{y}_{j,i}$ for P_j.

- Each party P_i outputs
$$\boldsymbol{y}_i = \sum_{j \neq i} \boldsymbol{y}_{i,j}.$$

[a] (i_1, j_1) may have a common element with (i_2, j_2). In this case, we can appropriately adapt the algorithm and instantiate the OT call via 3-LTMPC.

Figure 13: Semi-Honest n-LTMPC Protocol

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

OT Call: Running with parties $P_{i_1}, P_{j_1}, P_{i_2}, P_{j_2}$ and a simulator $\mathcal{S}$, $\mathcal{F}_{\mathsf{Int}}$ works as follows:

- Upon receiving input $(\mathsf{send}, P_{i_1}, \boldsymbol{x}_{i_1,j_1}, \boldsymbol{r}^{i_1}_{i_1,j_1}, \tau^{i_1}_{i_1,j_1})$ from party P_{i_1} and input $(\mathsf{send}, P_{j_1}, \boldsymbol{x}_{j_1,i_1}, \boldsymbol{r}^{j_1}_{i_1,j_1}, \tau^{j_1}_{i_1,j_1})$ from party P_{j_1}, notify $\mathcal{S}$ about the sender information.
- Compute (x_0, x_1) from $\boldsymbol{x}_{i_1,j_1} \parallel \boldsymbol{x}_{j_1,i_1}$, $\boldsymbol{r}_{i_1,j_1} \leftarrow \mathsf{Dec}_{\mathsf{BCL}}(\boldsymbol{r}^{i_1}_{i_1,j_1}, \boldsymbol{r}^{j_1}_{i_1,j_1})$ and $\tau_{i_1,j_1} \leftarrow \mathsf{Dec}_{\mathsf{BCL}}(\tau^{i_1}_{i_1,j_1}, \tau^{j_1}_{i_1,j_1})$.
- Upon receiving input $(\mathsf{receive}, P_{i_2}, \boldsymbol{x}_{i_2,j_2}, \boldsymbol{r}^{i_2}_{i_2,j_2}, \tau^{i_2}_{i_2,j_2})$ from party P_{i_2} and input $(\mathsf{receive}, P_{j_2}, \boldsymbol{x}_{j_2,i_2}, \boldsymbol{r}^{j_2}_{i_2,j_2}, \tau^{j_2}_{i_2,j_2})$ from party P_{j_2}, notify $\mathcal{S}$ about the receiver information.
- Compute $b \in \{0,1\}$ from $\boldsymbol{x}_{i_2,j_2} \parallel \boldsymbol{x}_{j_2,i_2}$, $\boldsymbol{r}_{i_2,j_2} \leftarrow \mathsf{Dec}_{\mathsf{BCL}}(\boldsymbol{r}^{i_2}_{i_2,j_2}, \boldsymbol{r}^{j_2}_{i_2,j_2})$ and $\tau_{i_2,j_2} \leftarrow \mathsf{Dec}_{\mathsf{BCL}}(\tau^{i_2}_{i_2,j_2}, \tau^{j_2}_{i_2,j_2})$.
- Sample a random 2-BCLSS of x_b, say $\langle x_b \rangle_2^{\mathsf{BCL}} = (x_b^{i_2}, x_b^{j_2})$. Once $\mathcal{S}$ acknowledges, output the value $x_b^{i_2}$ to P_{i_2} and $x_b^{j_2}$ to P_{j_2}.

Output Decoding: Running with parties P_i, P_j and a simulator $\mathcal{S}$, $\mathcal{F}_{\mathsf{Int}}$ works as follows:

- Upon receiving input $(\mathsf{send}, P_i, \boldsymbol{x}_{i,j}, \boldsymbol{r}^i_{i,j}, \tau^i_{i,j})$ from party P_i and input $(\mathsf{send}, P_j, \boldsymbol{x}_{j,i}, \boldsymbol{r}^j_{i,j}, \tau^j_{i,j})$ from party P_j, notify $\mathcal{S}$ about the computation.
- Decode $\boldsymbol{y}_{i,j} \parallel \boldsymbol{y}_{j,i}$ from $\boldsymbol{x}_{i,j} \parallel \boldsymbol{x}_{j,i}$, $\boldsymbol{r}_{i,j} \leftarrow \mathsf{Dec}_{\mathsf{BCL}}(\boldsymbol{r}^i_{i,j}, \boldsymbol{r}^j_{i,j})$ and $\tau_{i,j} \leftarrow \mathsf{Dec}_{\mathsf{BCL}}(\tau^i_{i,j}, \tau^j_{i,j})$.
- Once $\mathcal{S}$ acknowledges, output $\boldsymbol{y}_{i,j}$ to P_i and $\boldsymbol{y}_{j,i}$ to P_j.

Figure 14: The Inner Functionality

Hybrid Hyb_1. In this hybrid, we randomly sample an additive secret sharing $(\boldsymbol{x}_{i,j})_{j\neq i}$ of each input $\boldsymbol{x}_i$ and the 2-BCLSS of internal randomness $\langle \boldsymbol{r}_{i,j}\rangle_2^{\mathsf{BCL}}$ of every virtual party $V_{i,j}$, then emulate a real execution of the protocol π_{Ext} to attain the transcript as well as the output $\boldsymbol{y}_{i,j} \parallel \boldsymbol{y}_{j,i}$ of each $V_{i,j}$. Subsequently, we compute a random 2-BCLSS $\langle \tau_{i,j}\rangle_2^{\mathsf{BCL}}$ of each virtual party's transcript to fill in the view of every party P_i. Every oracle call to $\mathcal{F}_{\mathsf{Int}}$ is hardwired so that it constantly outputs the predetermined values $\langle x_b\rangle_2^{\mathsf{BCL}}$ or $\boldsymbol{y}_{i,j} \parallel \boldsymbol{y}_{j,i}$.

This modification does not change the distribution of any party's view, thus Real and Hyb_1 are identical.

Hybrid Hyb_2. In this hybrid, for every honest virtual party $V_{i,j}$ (namely, the virtual parties such that both P_i and P_j are honest), we switch the 2-BCLSSs $\langle \boldsymbol{r}_{i,j}\rangle_2^{\mathsf{BCL}}$ and $\langle \tau_{i,j}\rangle_2^{\mathsf{BCL}}$ to encode all-0 strings.

Since these 2-BCLSSs are never explicitly decoded, the security of the underlying 2-BCLSS scheme ensures that Hyb_1 and Hyb_2 are statistically indistinguishable under ℓ leakage.

Hybrid Ideal. This hybrid is the ideal world execution. We construct a simulator $\mathcal{S}$ of the protocol π_{LTMPC} as follows:

1. Compute the inputs and outputs of corrupted virtual parties:
   - For every corrupted party P_i, sample random additive secret sharings of input $\boldsymbol{x}_i$ and output $\boldsymbol{y}_i$ as $(\boldsymbol{x}_{i,j})_{j\neq i}$ and $(\boldsymbol{y}_{i,j})_{j\neq i}$.
   - For every honest party P_i, let P_{i^*} be another honest party and sample random strings as the shares $\boldsymbol{x}_{i,j}$ and $\boldsymbol{y}_{i,j}$ for all $P_j \notin \{P_i, P_{i^*}\}$, while the shares $\boldsymbol{x}_{i,i^*}$ and $\boldsymbol{y}_{i,i^*}$ can be extracted in the state translation function T_i.
2. Invoke a simulator $\mathcal{S}_{\mathsf{Ext}}$ of the semi-honest GMW protocol π_{Ext} to emulate the view of corrupted parties, and compute a 2-BCLSS of the simulated view accordingly.
3. Forge $\langle \boldsymbol{r}_{i,j}\rangle_2^{\mathsf{BCL}} = (\boldsymbol{r}_{i,j}^i, \boldsymbol{r}_{i,j}^j)$ and $\langle \tau_{i,j}\rangle_2^{\mathsf{BCL}} = (\tau_{i,j}^i, \tau_{i,j}^j)$ of honest virtual parties $V_{i,j}$ with random 2-BCLSSs of all-0 strings.
4. To output a state translation function T_i of an honest party P_i, first generate random additive sharings as $(\boldsymbol{x}_{i,j})_{j\neq i}$ and $(\boldsymbol{y}_{i,j})_{j\neq i}$ which are consistent with the ideal state $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ and the input-output pairs of corrupted virtual parties, then utilize the 2-BCLSSs prepared in Step 2 and Step 3 to complete P_i's view.

Observe that the only difference between Hyb_2 and Ideal is the view of corrupted virtual parties. The security of π_{Ext} guarantees that Hyb_2 and Ideal are statistically indistinguishable.

□

Putting Lemma 5.3, Lemma 5.4 and Lemma 5.2 together, we obtain the following theorem:

Theorem 5.5. *Let $\ell = \ell(\kappa)$ be a leakage bound. Then any adaptively well-formed n-party functionality $\mathcal{F}$ can be securely computed by an unconditionally secure semi-honest $(n, n-1, \ell)$-LTMPC protocol π_{LTMPC} in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model. Furthermore, if a semi-honest LTOT protocol ρ exists, the protocol $\pi_{\mathsf{LTMPC}}^{\rho}$ is a computationally secure semi-honest $(n, n-1, \ell)$-LTMPC of $\mathcal{F}$ in the plain model which makes only black-box use of ρ.*

6 Computationally Secure LTMPC Against Malicious Corruptions

6.1 Additional Preliminaries

Leakage-Tolerant Commit-and-Prove. Recall that in the standard (non-leakage-tolerant) model, the one-to-many commit-and-prove functionality $\mathcal{F}_{\mathsf{CP}}^{1:M}$ can be instantiated with a stand-alone perfectly binding non-interactive commitment scheme com together with an ideal zero-knowledge functionality $\mathcal{F}_{\mathsf{ZK}}$. The same construction is also applied to LTMPC, except for replacing com by a non-interactive equivocal commitment (for instance, the Feige-Shamir commitment [FS89, CLOS02] in the CRS model, which can be built from the commitment scheme of Naor [Nao91] with any one-way function). Note that [BCH12] has constructed an ideal leakage-tolerant zero-knowledge proof functionality based on the DDH assumption, we can therefore realize the leakage-tolerant commit-and-prove functionality $\mathcal{F}_{\mathsf{CP}}^{1:M}$ from the DDH assumption, assuming a common reference string and an authenticated broadcast channel.

Functionality $\mathcal{F}_{\mathsf{CP}}^{1:M}$

Running with parties $P_1, \ldots, P_n$ and a simulator $\mathcal{S}$, $\mathcal{F}_{\mathsf{CP}}^{1:M}$ (parameterized by a relation R) proceeds as follows:

Commit Phase:

- Upon receiving $(\mathsf{commit}, P_i, w)$ from any party P_i, append the string w to the list $\overline{w}$ of P_i (initially empty) and broadcast $(\mathsf{receipt}, P_i)$ to every party and $\mathcal{S}$.

Prove Phase:

- Upon receiving (prove, P_i, x) from any party P_i, verify the relation $R(x, \overline{w})$.
- If $R(x, \overline{w}) = 1$, broadcast (proof, P_i, x) to every party and $\mathcal{S}$.

Figure 15: The One-to-Many Commit-and-Prove Functionality

Semi-Malicious LTOT. [BGJ+13] designs a semi-malicious LTOT scheme in the CRS model. Their construction makes use of the DDH assumption and strengthened lossy trapdoor functions. Here we take a closer look at their construction. From a technical perspective, the necessity of lossy trapdoor functions is due to the need to ensure the following two conditions:

1. A corrupted sender-receiver pair cannot jointly select randomness in order to yield incorrect output.
2. A corrupted sender S cannot forge in an LTOT execution to let an honest receiver R learn incorrect output.

Our construction no longer requires the first condition (namely, we permit a corrupted sender-receiver pair to forge in the LTOT). Moreover, as we allow only bounded leakage, the necessity of using lossy trapdoor functions to prevent the second condition vanishes. To see this, we give a quick recap of their scheme:

- The CRS contains two fixed random messages $M_0, M_1 \xleftarrow{\$} M$ where M is a message space of exponential size, say $2^{\ell+\omega(\log\kappa)}$.
- Let k be a sufficiently large parameter. On input $b \in \{0,1\}$, the receiver R first generates a random subset $I \subset [4k]$ with size k. Then for each $i \in \{0,1\}$ and $j \in [4k]$, R honestly prepares $(\mathsf{pk}_{i,j}, \mathsf{sk}_{i,j})$ of a specific DDH-based public-key encryption scheme only if $i = b$ and $j \in I$; otherwise, it invokes an oblivious public-key generation algorithm oGen to sample a dummy public key.
- On input (x_0, x_1), the sender S also generates two random subsets $J_0, J_1 \subset [4k]$ with size k in advance. Upon receiving public keys $\{\mathsf{pk}_{i,j}\}_{i\in\{0,1\},j\in[4k]}$ from R, S honestly encrypts the corresponding plaintext $c_{i,j} \leftarrow \mathsf{Enc}(\mathsf{pk}_{i,j}, M_{x_i})$ using $\mathsf{pk}_{i,j}$ only if $j \in J_i$; otherwise, it invokes the oblivious ciphertext generation algorithm oEnc to sample $c_{i,j}$. The algorithm oEnc uniformly generates ciphertexts of a random message $m \xleftarrow{\$} M$. Hence, with overwhelming probability an oblivious $c_{i,j}$ will encrypt a value coinciding with neither M_0 nor M_1.
- R decrypts the ciphertexts $\{c_{b,j}\}_{j\in I}$. If there exists some $c_{b,j}$ that decrypts to M_u and no other $c_{b,j}$ decrypts to M_{1-u}, R outputs $x_b = u$. Otherwise, it outputs $\perp$.

A rushing adversary may control the sender S to choose some $J_b \cap I = \emptyset$, thus the honest receiver R will fail to decrypt. However, if we pick $k = \omega(\ell + \log\kappa)$ and require S to commit its random choice before seeing public keys, a computational leveraging argument directly shows that this bad event will occur with probability $2^\ell \exp(-\Omega(k)) = \mathsf{negl}(\kappa)$. Without any leakage information, a corrupted sender S can choose $J_b \cap I = \emptyset$ with only a probability of $\exp(-\Omega(k))$. Receiving ℓ-bit leakage in advance only makes the probability 2^ℓ times larger.

A more subtle issue is that the adversary may not follow the protocol to generate oblivious ciphertexts. Indeed, it can choose $i^* \in \{0,1\}$ and generate $c_{i^*,j} \leftarrow \mathsf{Enc}(\mathsf{pk}_{i^*,j}, M_{1-x_{i^*}})$ for $j \notin J_{i^*}$ (then explains $c_{i^*,j}$ as outputs of oEnc). This incurs a selective abort attack where an honest receiver only aborts if $b = i^*$. We use a coin-tossing protocol to avoid this issue. After sending the public keys $\{\mathsf{pk}_{i,j}\}$, the receiver R first invokes $\mathcal{F}_{\mathsf{CP}}^{1:M}$ to commit random strings $\{\boldsymbol{r}_{i,j}\}_{i\in\{0,1\},j\in[4k]}$. Then the sender S also commits random strings $\{\boldsymbol{r}'_{i,j}\}_{i\in\{0,1\},j\in[4k]}$. Subsequently, the strings $\boldsymbol{r}_{i,j}$ are sent to S, and S is required to use $\mathsf{oEnc}(\boldsymbol{r}_{i,j} + \boldsymbol{r}'_{i,j})$ to generate oblivious ciphertexts.

When S is honest but R is corrupted, the simulator can decommit $\boldsymbol{r}_{i,j}$ before choosing $\boldsymbol{r}'_{i,j}$, thus does not affect the original analysis. Conversely, when R is honest but S is corrupted, the adversary is only able to generate an incorrect ciphertext of $M_{1-x_{i^*}}$ with probability $2^\ell/|M| = \mathsf{negl}(\kappa)$, as oEnc samples a uniform ciphertext [8].

6.2 The External Protocol

In this section, we elaborate on how to realize the sketch of multi-party malicious LTMPC presented in Section 2.2. The first step is to implement an m-party protocol π_{Ext} of *deterministic* functionality $\mathcal{F}$ in the broadcast channel model $\mathcal{F}_{\mathsf{BC}}$ with the following properties:

1. π_{Ext} is a non-leakage-tolerant MPC protocol of $\mathcal{F}$ against $t \le m-1$ malicious corruptions.
2. π_{Ext} is a leakage-tolerant MPC protocol of $\mathcal{F}$ against exactly $t = m-1$ malicious corruptions.
3. The outputs of π_{Ext} are consistent with $\mathcal{F}$ even for corrupted parties. That is, every party P_i needs to provide a certificate of its output $\boldsymbol{y}_i$.

[8] This coin-tossing protocol does not deviate from the impossibility result Lemma A.1. The randomness $\boldsymbol{r}_{i,j} + \boldsymbol{r}'_{i,j}$ is not required to be uniformly random, but suffices to lie outside an exponentially small subset.

For this purpose, we apply the offline-online MPC protocol of [BDOZ11] but drop the use of MACs, because we prevent all illegitimate behaviors via the commit-and-prove functionality $\mathcal{F}_{\mathsf{CP}}^{1:M}$. To produce a Beaver triple $(\langle a\rangle_m, \langle b\rangle_m, \langle c\rangle_m)$, each party locally generates $(\langle a\rangle_m, \langle b\rangle_m)$ and computes $\langle c\rangle_m = \langle ab\rangle_m$ via pairwise LTOTs. However, as we do not assume the perfect correctness of the underlying LTOT scheme, a corrupted sender-receiver pair may collude to cheat in the LTOT leading to an incorrect triple. Hence, we verify the correctness of $(\langle a\rangle_m, \langle b\rangle_m, \langle c\rangle_m)$ by sacrificing another triple $(\langle f\rangle_m, \langle g\rangle_m, \langle h\rangle_m)$. Here, the additive sharings $\langle f\rangle_m$ and $\langle h\rangle_m$ are defined over a sufficiently large extension field $\mathbb{F} = \mathbb{F}_{2^k}$ where $k = \ell + \omega(\log \kappa)$, denoted by notation $\langle\cdot\rangle_m^{\mathbb{F}}$.

Protocol π_{Ext}

$P_1, \ldots, P_m$ evaluate the m-party function $\mathcal{F}$. Every party commits its input, output as well as internal randomness to $\mathcal{F}_{\mathsf{CP}}^{1:M}$, and proves compliance with the protocol.

- For each input wire $\boldsymbol{x}_i$ carrying an input bit of party P_i, P_i samples a random sharing $\langle \boldsymbol{x}_i\rangle_m$ and broadcasts the shares $\langle \boldsymbol{x}_i\rangle_{m,[m]/\{i\}}$ of other parties.
- For each addition gate of $\mathcal{F}$ with input wires α, β and output wire γ, the parties locally add $\langle\alpha\rangle_m$ and $\langle\beta\rangle_m$ to obtain $\langle\gamma\rangle_m$.
- For each multiplication gate of $\mathcal{F}$ with input wires α, β and output wire γ:
  - The parties invoke protocol π_{Triple} to prepare a random Beaver triple $(\langle a\rangle_m, \langle b\rangle_m, \langle c\rangle_m)$.
  - The parties open $\langle\alpha\rangle_m - \langle a\rangle_m$ to obtain ϵ, and open $\langle\beta\rangle_m - \langle b\rangle_m$ to obtain δ.
  - Each party locally computes its share of
    $$\langle\gamma\rangle_m = \epsilon\delta + \epsilon\langle b\rangle_m + \langle a\rangle_m\delta + \langle c\rangle_m.$$
    Here, in order to add the public constant value $\epsilon\delta$, we simply let P_1 add $\epsilon\delta$ to its share $\langle\gamma\rangle_{m,1}$.
- For each output wire $\boldsymbol{y}_i$ carrying an output bit of party P_i, each party $P_j \neq P_i$ broadcasts its share of $\boldsymbol{y}_i$ so that P_i can reconstruct $\boldsymbol{y}_i$.

Figure 16: External Protocol

Security. We now prove that the protocol π_{Ext} satisfies the three properties as desired. Here we only provide the proof of the leakage-tolerance property (Property 2), while Property 1 follows from a standard MPC security argument. Assuming that P_i is the unique honest party, we present a leakage-oblivious simulator $\mathcal{S}$ as follows:

- For each input wire $\boldsymbol{x}_j$ of a corrupted party $P_j \neq P_i$, $\mathcal{S}$ receives the sharing $\langle \boldsymbol{x}_j\rangle_m$ from $\mathcal{F}_{\mathsf{CP}}^{1:M}$ and submits $\boldsymbol{x}_j$ to the ideal functionality as the inferred input of P_j; and for each input wire $\boldsymbol{x}_i$ of P_i, $\mathcal{S}$ randomly generates the shares of corrupted parties, and computes the share of P_i in the state translation function T_i accordingly.
- $\mathcal{S}$ honestly emulates each addition gate.
- For each multiplication gate, $\mathcal{S}$ generates $\langle\epsilon\rangle_{m,i}, \langle\delta\rangle_{m,i} \xleftarrow{\$} \mathbb{F}_2$, then emulates the protocol π_{Triple}:

Protocol π_{Triple}

Every party commits its input, output as well as internal randomness to $\mathcal{F}_{\mathsf{CP}}^{1:M}$, and proves compliance with the protocol.

- The parties prepare a random sharing $\langle e \rangle_m^{\mathbb{F}}$ over $\mathbb{F}$.
- Each party locally samples the shares of four additive sharings $(\langle a \rangle_m, \langle b \rangle_m, \langle f \rangle_m^{\mathbb{F}}, \langle g \rangle_m)$.
- Every ordered pair of parties P_i and P_j performs two invocations of the semi-malicious LTOT to prepare

$$c_{i,j}^i + c_{i,j}^j = \langle a \rangle_{m,i} \cdot \langle b \rangle_{m,j},$$
$$h_{i,j}^i + h_{i,j}^j = \langle f \rangle_{m,i}^{\mathbb{F}} \cdot \langle g \rangle_{m,j}.$$

  For more details, P_i randomly generates $c_{i,j}^i \xleftarrow{\$} \mathbb{F}_2$ (resp. $h_{i,j}^i \xleftarrow{\$} \mathbb{F}$), then runs a (string) 2-choose-1 LTOT on input $(-c_{i,j}^i, \langle a \rangle_{m,i} - c_{i,j}^i)$ (resp. $(-h_{i,j}^i, \langle f \rangle_{m,i}^{\mathbb{F}} - h_{i,j}^i)$) to let P_j learn $c_{i,j}^j$ (resp. $h_{i,j}^j$).
- Each party P_i locally computes $\langle c \rangle_{m,i} = \langle a \rangle_{m,i} \langle b \rangle_{m,i} + \sum_{j \neq i} (c_{i,j}^i + c_{j,i}^i)$ as well as $\langle h \rangle_{m,i}^{\mathbb{F}} = \langle f \rangle_{m,i}^{\mathbb{F}} \langle g \rangle_{m,i} + \sum_{j \neq i} (h_{i,j}^i + h_{j,i}^i)$.
- The parties open $\langle e \rangle_m^{\mathbb{F}}$ to obtain the challenge $e \in \mathbb{F}$.
- The parties open $e\langle a \rangle_m - \langle f \rangle_m^{\mathbb{F}}$ to obtain $\rho \in \mathbb{F}$, and open $\langle b \rangle_m - \langle g \rangle_m$ to obtain $\sigma \in \mathbb{F}_2$. Then they open

$$\langle \tau \rangle_m^{\mathbb{F}} = e\langle c \rangle_m - \langle h \rangle_m^{\mathbb{F}} - \rho\sigma - \rho\langle g \rangle_m - \langle f \rangle_m^{\mathbb{F}} \sigma$$

  and verify that the difference τ is 0.
- The output of party P_i is $(\langle a \rangle_{m,i}, \langle b \rangle_{m,i}, \langle c \rangle_{m,i})$.

Figure 17: Triple Generation Protocol

– $\mathcal{S}$ generates $\langle e\rangle^{\mathbb{F}}_{m,i} \xleftarrow{\$} \mathbb{F}$ and receives the corrupted parties' shares of $\langle e\rangle^{\mathbb{F}}_{m}$ from $\mathcal{F}^{1:M}_{\mathsf{CP}}$.

– $\mathcal{S}$ randomly samples $\langle \rho\rangle_{m,i} \xleftarrow{\$} \mathbb{F}$ and $\langle \sigma\rangle_{m,i} \xleftarrow{\$} \mathbb{F}_2$, then in T_i it assigns

$$\begin{aligned}\langle a\rangle_{m,i} &= \langle \alpha\rangle_{m,i} - \langle \epsilon\rangle_{m,i},\\ \langle b\rangle_{m,i} &= \langle \beta\rangle_{m,i} - \langle \delta\rangle_{m,i},\\ \langle f\rangle^{\mathbb{F}}_{m,i} &= e\langle a\rangle_{m,i} - \langle \rho\rangle^{\mathbb{F}}_{m,i},\\ \langle g\rangle_{m,i} &= \langle b\rangle_{m,i} - \langle \sigma\rangle_{m,i}.\end{aligned}$$

After that, it receives the corresponding shares of corrupted parties from $\mathcal{F}^{1:M}_{\mathsf{CP}}$.

– $\mathcal{S}$ honestly emulates the (possibly cheating) execution of LTOTs between two corrupted parties. Moreover, for every LTOT between P_i and some corrupted party P_j, $\mathcal{S}$ prepares random shares $(c^j_{i,j}, h^j_{i,j}) \xleftarrow{\$} \mathbb{F}_2 \times \mathbb{F}$ of P_j and invokes the simulator of LTOT. $\mathcal{S}$ then receives the corrupted parties' shares of $\langle c\rangle_m$ and $\langle h\rangle^{\mathbb{F}}_m$ from $\mathcal{F}^{1:M}_{\mathsf{CP}}$.

– $\mathcal{S}$ then opens $\langle \tau\rangle^{\mathbb{F}}_m$ and emulates the check honestly. To do so, we only need to explain how $\mathcal{S}$ computes $\langle \tau\rangle^{\mathbb{F}}_{m,i}$. Note that the adversary can only forge in LTOTs between two corrupted parties, therefore $\mathcal{S}$ can compute the *ideal* values of corrupted parties' shares $(\langle \tau\rangle^{\mathbb{F}}_{m,j,\mathsf{Ideal}})_{j\neq i}$ when the adversary does not forge in LTOTs, since it knows the accurate shares $\{\langle a\rangle_{m,j}, \langle b\rangle_{m,j}, \langle f\rangle^{\mathbb{F}}_{m,j}, \langle g\rangle_{m,j}\}_{j\neq i}$ of corrupted parties. In this way, it can recover $\tau^{\mathbb{F}}_{m,i} = -\sum_{j\neq i}\langle \tau\rangle^{\mathbb{F}}_{m,j,\mathsf{Ideal}}$.

Finally, $\mathcal{S}$ simulates $\langle \gamma\rangle_m$ based on the above information.

- For each output wire $\boldsymbol{y}_j$ of corrupted party P_j, $\mathcal{S}$ computes the share of P_i as $\boldsymbol{y}_j - \sum_{k\neq i}\langle \boldsymbol{y}_j\rangle_{m,k}$; and for each output wire $\boldsymbol{y}_i$, $\mathcal{S}$ simply receives the shares of corrupted parties from $\mathcal{F}^{1:M}_{\mathsf{CP}}$.

It is not hard to see that $\mathcal{S}$ perfectly emulates the execution of π_{Ext}. Furthermore, we claim that the protocol π_{Triple} outputs a Beaver triple with overwhelming probability when the protocol does not abort, and consequently Property 3 of π_{Ext} is ensured. This can be argued by a computational leveraging argument: the adversary can cause a faulty triple to pass the test with a probability of only $1/|\mathbb{F}|$ if it does not gain any leakage from the honest party P_i, and obtaining at most ℓ bits of leakage *before opening* $\langle e\rangle^{\mathbb{F}}_m$ only increases this probability to $2^\ell/|\mathbb{F}| = \mathsf{negl}(\kappa)$.

6.3 The Full Scheme

We then adopt the party virtualization approach to compile the external protocol π_{Ext} into a maliciously secure n-party LTMPC π_{LTMPC}. Concretely, the protocol π_{Ext} contains $n(n-1)/2$ virtual parties $\{V_{i,j}\}$ and n actual parties $\{V_i\}$. The input and output of a party P_i are shared by $\{V_{i,j}\}_{j\neq i} \cup \{V_i\}$. Each virtual party $V_{i,j}$ is mimicked by a *semi-honest* 2-LTMPC between P_i and P_j, while the internal randomness of $V_{i,j}$ is represented by a 2-BCLSS. Recall that the protocol π_{Ext} is designed in the broadcast channel model. To emulate a broadcast channel of a virtual party $V_{i,j}$, we simply let P_i and P_j run a 2-LTMPC, then P_i will broadcast the resulting message m.

Security. We exploit the security of the external protocol π_{Ext} to prove the security of π_{LTMPC} via a hybrid argument. The corruption set of π_{Ext} is determined based on the corruption threshold t:

1. $t \leq n-2$**:** We mark all $\{V_i\}_{i\in[n]}$ as corrupted; and a 2-ary virtual party $V_{i,j}$ is considered honest if both P_i and P_j are honest.

Protocol π_{LTMPC}

$P_1, \ldots, P_n$ evaluate the n-party function $\mathcal{F}$. Let com be a non-interactive equivocal commitment scheme.

- Construct the external protocol π_{Ext} over $m = n(n+1)/2$ virtual parties $\{V_{i,j} = V_{j,i} : 1 \le i < j \le n\} \cup \{V_i : i \in [n]\}$ in the broadcast channel model, which computes the deterministic functionality $\mathcal{F}'$ defined as follows:

  1. On input $\boldsymbol{x}_{i,j} \parallel rx_{i,j} \parallel \boldsymbol{x}_{j,i} \parallel rx_{j,i}$ of each virtual party $V_{i,j}$ and input $\boldsymbol{x}_{i,i} \parallel rx_{i,i}$ of each V_i, $\mathcal{F}'$ first verifies that for every $i, j \in [n]$, $\mathsf{com}(\boldsymbol{x}_{i,j}; rx_{i,j}) = cx_{i,j}$, where $cx_{i,j}$ is a common commitment value defined later. Analogously, on input $\boldsymbol{y}_{i,j} \parallel ry_{i,j} \parallel \boldsymbol{y}_{j,i} \parallel ry_{j,i}$ of each virtual party $V_{i,j}$, it also verifies that for every $i \neq j$, $\mathsf{com}(\boldsymbol{y}_{i,j}; ry_{i,j}) = cy_{i,j}$, where $cy_{i,j}$ is a common commitment value defined later.
  2. $\mathcal{F}'$ extracts the input $\boldsymbol{x}_i$ of each party P_i:
  $$\boldsymbol{x}_i = \sum_{j=1}^{n} \boldsymbol{x}_{i,j}.$$
  3. $\mathcal{F}'$ invokes the functionality $\mathcal{F}$ to compute the output $\boldsymbol{y}_i$ of P_i.
  4. Then, $\mathcal{F}'$ computes the output $\boldsymbol{y}_{i,i}$ of each V_i:
  $$\boldsymbol{y}_{i,i} = \boldsymbol{y}_i - \sum_{j \neq i} \boldsymbol{y}_{i,j}.$$

- Each party P_i uses $\mathcal{F}_{\mathsf{CP}}^{1:M}$ to commit its input $\boldsymbol{x}_i$ and a random additive sharing $(\boldsymbol{x}_{i,j})_{j \in [n]}$. It also announces the commitments $cx_{i,j} \leftarrow \mathsf{com}(\boldsymbol{x}_{i,j}; rx_{i,j})$ and proves the consistency with $\boldsymbol{x}_i$ via $\mathcal{F}_{\mathsf{CP}}^{1:M}$.
- Then each party P_i also generates random shares $(\boldsymbol{y}_{i,j})_{j \neq i}$ and announces commitments $cy_{i,j} \leftarrow \mathsf{com}(\boldsymbol{y}_{i,j}; ry_{i,j})$ analogously.
- Each pair of parties P_i and P_j independently samples shares $\boldsymbol{r}_{i,j}^i$ and $\boldsymbol{r}_{i,j}^j$ as a 2-BCLSS $\langle \boldsymbol{r}_{i,j} \rangle_2^{\mathsf{BCL}} = (\boldsymbol{r}_{i,j}^i, \boldsymbol{r}_{i,j}^j)$ of the internal randomness of $V_{i,j}$.
- $P_1, \ldots, P_n$ emulate the protocol π_{Ext} in which the transcript is broadcast to every party. In each instruction of π_{Ext}, either some V_i or some $V_{i,j}$ broadcasts a message. For the former request, we simply let P_i compute and broadcast the message; and for the latter one, P_i and P_j will run a semi-honest 2-LTMPC (which is designed under the secure point-to-point channel model) to let P_i learn the message via functionality $\mathcal{F}_{\mathsf{Int}}$, then P_i broadcasts the message.
- In this way, each party P_i will learn the output $\boldsymbol{y}_{i,i}$ of V_i, then it outputs
$$\boldsymbol{y}_i = \sum_{j=1}^{n} \boldsymbol{y}_{i,j}.$$

Figure 18: Malicious n-LTMPC Protocol

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

Running with parties P_i, P_j and a simulator $\mathcal{S}$, $\mathcal{F}_{\mathsf{Int}}$ works as follows:

- Upon receiving input $(\mathsf{send}, P_i, \boldsymbol{x}_{i,j} \parallel rx_{i,j}, \boldsymbol{y}_{i,j} \parallel ry_{i,j}, \boldsymbol{r}^i_{i,j})$ from party P_i and input $(\mathsf{send}, P_j, \boldsymbol{x}_{j,i} \parallel rx_{j,i}, \boldsymbol{y}_{j,i} \parallel ry_{j,i}, \boldsymbol{r}^j_{i,j})$ from party P_j, notify $\mathcal{S}$ about the computation.
- Compute the next message m from $\boldsymbol{x}_{i,j} \parallel rx_{i,j} \parallel \boldsymbol{x}_{j,i} \parallel rx_{j,i}$, $\boldsymbol{y}_{i,j} \parallel ry_{i,j} \parallel \boldsymbol{y}_{j,i} \parallel ry_{j,i}$, $\boldsymbol{r}_{i,j} \leftarrow \mathsf{Dec}_{\mathsf{BCL}}(\boldsymbol{r}^i_{i,j}, \boldsymbol{r}^j_{i,j})$ and the common transcript.
- Once $\mathcal{S}$ acknowledges, send m to P_i.

Figure 19: The Inner Functionality

2. $t = n - 1$**:** Let P_i be the unique honest party. We mark all virtual parties except for V_i as corrupted. Note that in this case, even the randomness of V_i is under leakage attack.

Hybrid Real**.** This hybrid is the real-life execution.

Hybrid Hyb_1**.** In this hybrid, we simulate the joint view of every party in Real according to the inputs of all honest parties as follows:

- Determine the input sharing $\langle \boldsymbol{x}_i \rangle_n$ of each corrupted party P_i from $\mathcal{F}_{\mathsf{CP}}^{1:M}$ and receive the commitments $cx_{i,\cdot}, cy_{i,\cdot}$. Then we honestly prepare the input sharings $(\boldsymbol{x}_{i,j})_{j\in[n]}$ and output sharings $(\boldsymbol{y}_{i,j})_{j\neq i}$ of each honest party P_i, as well as the corresponding commitments.
- Emulate an honest execution of protocol π_{Ext}. Specifically, for each broadcast message m of a virtual party $V_{i,j}$: if $V_{i,j}$ is corrupted, we honestly simulate the 2-LTMPC; otherwise, we first compute the value of m, then invoke a simulator $S_{i,j}$ on input m to simulate the underlying 2-LTMPC.
- Honestly emulate the output decoding phase to obtain each output $\boldsymbol{y}_i$.

  By the security of the underlying semi-honest 2-LTMPC, we have $\mathsf{Real} \approx_C \mathsf{Hyb}_1$.

Hybrid Hyb_2**.** In this hybrid, we hardwire the output m to each simulator $S_{i,j}$ of honest virtual parties, then switch the 2-BCLSS $\langle \boldsymbol{r}_{i,j} \rangle_2^{\mathsf{BCL}}$ to encode all-0 strings.

The indistinguishability between Hyb_1 and Hyb_2 is ensured by the security of 2-BCLSS.

Hybrid Hyb_3**.** In this hybrid, we invoke the simulator S_{Ext} of protocol π_{Ext} to emulate the view of corrupted parties, where the ideal functionality $\mathcal{F}'$ can be emulated by an oracle call to $\mathcal{F}$. Specifically:

1. $t \leq n - 2$**:** we simulate the messages broadcast from honest virtual parties using S_{Ext}.
2. $t = n - 1$**:** we exploit S_{Ext} to simulate the transcript of the unique honest party.

In the two cases, we make use of Property 1 or Property 2 of π_{Ext}, respectively, to show the indistinguishability between Hyb_2 and Hyb_3.

Hybrid Hyb_4. In this hybrid, before emulating Step 3 of $\mathcal{F}'$, we additionally verify that for each pair comprised of an honest party P_i and a corrupted party P_j, the (corrupted) virtual party $V_{i,j}$ submits the same input share $\boldsymbol{x}_{i,j}$ and output share $\boldsymbol{y}_{i,j}$ as we prepared before. In this way, the decoded input of each honest party P_i must be identical to $\boldsymbol{x}_i$.

We can see that Hyb_3 and Hyb_4 are computationally indistinguishable, since the binding property of the underlying commitment scheme com ensures that the adversary can only provide a different opening of some $cx_{i,j}$ or $cy_{i,j}$ with negligible probability.

Hybrid Ideal. This hybrid is the ideal world distribution. We construct a simulator $\mathcal{S}$ as follows:

1. $\mathcal{S}$ receives the input sharings $\langle \boldsymbol{x}_i \rangle_n$ of every corrupted party from $\mathcal{F}_{\mathsf{CP}}^{1:M}$ and submits them to $\mathcal{F}$. In this way, it can obtain the ideal functionality outputs of all corrupted parties, while each state translation function T_i of an honest P_i will receive the ideal output $\boldsymbol{y}_i$.

2. For each honest party P_i, $\mathcal{S}$ randomly samples the input share $\boldsymbol{x}_{i,j}$ and output share $\boldsymbol{y}_{i,j}$ of each corrupted virtual parties $V_{i,j}$ (and if $t < n-1$, also the shares $\boldsymbol{x}_{i,i}$ and $\boldsymbol{y}_{i,i}$ of V_i); it then generates the remaining shares in T_i. Then $\mathcal{S}$ completes the sharings depending on the private input-output pair $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ in T_i.

   Then, $\mathcal{S}$ exploits the equivocal property of com to obliviously prepare the commitments $\{cx_{i,\cdot}\}, \{cy_{i,\cdot}\}$, and computes the corresponding decommitments in T_i.

3. $\mathcal{S}$ invokes the sub-simulators S_{Ext} and $S_{i,j}$ to emulate the protocol π_{Ext}, where the oracle call to $\mathcal{F}$ is answered using the outputs obtained in Step 1. $\mathcal{S}$ aborts if some $\boldsymbol{x}_{i,j}, \boldsymbol{y}_{i,j}$ of a corrupted virtual party $V_{i,j}$ (or V_i if $i = j$) provided to $\mathcal{F}'$ is inconsistent with the recorded value.

4. $\mathcal{S}$ honestly emulates the output decoding phase.

Note that Hyb_4 and Ideal may differ only in the generation of commitments of honest parties. Therefore, we have $\mathsf{Hyb}_4 \approx_C \mathsf{Ideal}$ by the equivocal property of com.

We conclude with the following theorem:

Theorem 6.1. *Let $\ell = \ell(\kappa)$ be a leakage bound. Suppose the DDH assumption holds. Then for any $n \geq 2$, every n-party deterministic functionality $\mathcal{F}$ admits a computationally secure malicious $(n, n-1, \ell)$-LTMPC with abort implementation in the CRS model.*

Remark 6.2. The technique used in Theorem 6.1 can also be applied to the proof of Theorem 5.5, recovering a direct reduction from semi-honest n-LTMPC to 2-LTMPC for every $n \geq 3$. However, as a trade-off, the reduction we obtained in this way is no longer black-box in the LTOT.

7 Unconditionally Secure Leakage-Tolerant Computation

7.1 Barriers Toward Unconditionally Secure LTMPC

We first show a somewhat unexpected negative result: it is impossible to realize statistically secure LTMPC in the plain model, even if there is no corrupted party and the scheme tolerates negligible correctness error!

Theorem 7.1. *For any $n \geq 2$, there exists some n-party functionality $\mathcal{F}$, such that no statistically secure $(n, 0, 2)$-LTMPC protocol securely computes $\mathcal{F}$ in the plain model.*

Proof. Consider the following n-party functionality $\mathcal{F}$: party P_1 takes input bit x_1 and party P_2 takes input bit x_2, while P_1 is willing to compute $y = x_1 \wedge x_2$ (the ideal inputs and outputs of the parties $P_3, \ldots, P_n$ are simply set to $\perp$). Suppose that there exists an $(n, 0, 2)$-LTMPC protocol computing functionality $\mathcal{F}$. We consider the transcript τ of P_1.

Denote the random variable of transcript τ w.r.t. input (x_1, x_2) in a real-life execution as $\tau(x_1, x_2)$. We first claim that the distributions of $\tau(0,0)$ and $\tau(0,1)$ are statistically close. To argue for this, we observe that applying a 1-bit leakage to P_1 suffices to distinguish any statistically distant distributions of τ. As a consequence, the real-life distribution $\tau(x_1, x_2)$ is always statistically close to the ideal world distribution output by the state translation function T_1, so we have $\tau(0,0) \approx_S \tau(0,1)$.

Moreover, in order to verify the consistency of the transcripts $\tau_{1,i}$ and $\tau_{i,1}$, we only need to apply 1-bit leakage to both P_1 and P_i: let r be an independently chosen random string, the leakage queries $\langle \tau_{1,i}, r \rangle$ and $\langle \tau_{i,1}, r \rangle$ are identical with probability only $1/2$ if $\tau_{1,i} \neq \tau_{i,1}$. Hence, if the adversary randomly picks an index $i \in [2, n]$ and conducts the above test, with overwhelming probability the simulated transcript of P_1 must be simultaneously consistent with $(P_2, \ldots, P_n)$. Since the ideal states of $P_2, \ldots, P_n$ are identical between $(x_1, x_2) = (0,0)$ and $(x_1, x_2) = (1,0)$, the transcripts $\tau(0,0)$ and $\tau(1,0)$ should also be statistically close to each other. We conclude that $\tau(0,0) \approx_S \tau(0,1) \approx_S \tau(1,0)$.

We then apply the well-known impossibility result of information-theoretic two-party computation. We use the shorthand x^+ to denote $\max(x, 0)$ for any real number x. Then, we show that the transcript $\tau(1,1)$ is also statistically close to $\tau(0,0)$ by directly computing the statistical distance:

$$
\begin{aligned}
&\Delta(\tau(0,0), \tau(1,1)) \\
=&\mathbb{E}_{\tau \leftarrow \tau(0,0)} \left[1 - \frac{\Pr[\tau(1,1) = \tau]}{\Pr[\tau(0,0) = \tau]} \right]^+ \\
=&\mathbb{E}_{\tau \leftarrow \tau(0,0)} \left[1 - \frac{\Pr[\tau(0,1) = \tau] \cdot \Pr[\tau(1,0) = \tau]}{\Pr[\tau(0,0) = \tau]^2} \right]^+ \\
\leq&\mathbb{E}_{\tau \leftarrow \tau(0,0)} \left[1 - \frac{\Pr[\tau(0,1) = \tau]}{\Pr[\tau(0,0) = \tau]} \right]^+ + \left[1 - \frac{\Pr[\tau(1,0) = \tau]}{\Pr[\tau(0,0) = \tau]} \right]^+ \\
=&\Delta(\tau(0,0), \tau(0,1)) + \Delta(\tau(0,0), \tau(1,0)) \\
=&\mathsf{negl}(\kappa),
\end{aligned}
$$

where the third line follows from the property of interactive protocols, and the fourth line exploits the following elementary fact

- For all $u, v \geq 0$, we have $(1 - uv)^+ \leq (1 - u)^+ + (1 - v)^+$.

Putting the above results together, we conclude that $\tau(1,0) \approx_S \tau(1,1)$, which contradicts the (imperfect) correctness of the LTMPC scheme. □

7.2 $(2,0)$-LTMPC in Presence of Erasure

In this section, we discuss the construction of $(2,0)$-LTMPC in the erasure model. Let $\ell = \ell(\kappa)$ be a leakage bound. The first step is to establish a $(2,0)$-LTOT protocol $\mathsf{ErasureOT}$:

Security. The security proof essentially mimics the argument of [CDSMW09]. Formally, we construct a simulator $\mathcal{S}$ of the protocol $\mathsf{ErasureOT}$ that outputs two state translation functions T_S and T_R. Then we use a hybrid argument to show that no adversary can distinguish (T_S, T_R) from a real-life execution using only BCL leakage.

Protocol $\mathsf{ErasureOT}$

On input (x_0, x_1) and input b, sender S and receiver R evaluate the $\mathcal{F}_{\mathsf{OT}}$ functionality. For each $i \in \{0,1\}$:

1. Let $\lambda = 10\max\{\kappa, \ell\}$ and $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{oEnc})$ be an LROTP scheme with parameter λ and message length κ. The receiver R generates 4κ LROTP keys $\mathsf{key}_1^i, \ldots, \mathsf{key}_{4\kappa}^i \in \mathbb{F}_2^\lambda$ such that $\mathsf{key}_j^i \leftarrow \mathsf{Gen}$, and reveals the keys to S.
2. If $i = b$, R samples a uniformly random subset $I \subset [4\kappa]$ of size κ, and erases all $\{\mathsf{key}_j^i\}_{j \notin I}$ from its view; otherwise, R simply erases all the keys.
3. S also samples a uniformly random subset $J^i \subset [4\kappa]$ of size κ, then erases all $\{\mathsf{key}_j^i\}_{j \notin J^i}$ from its view.
4. Once the erasure operations of both parties are completed, S generates a ciphertext $M_j^i \in \mathbb{F}_2^{\lambda \times \kappa}$ for every index $j \in [4\kappa]$ and sends them to R. For $j \in J^i$, M_j^i is computed as an encryption of $(x_i)^\kappa$: $M_j^i \leftarrow \mathsf{Enc}(\mathsf{key}_j^i, (x_i)^\kappa)$; for $j \notin J^i$, M_j^i is simply produced by the oblivious ciphertext generation algorithm oEnc.
5. If $i = b$, R aims to extract the output x_b:
   - For each $j \in I$, decrypts $\boldsymbol{u}_j \leftarrow \mathsf{Dec}(\mathsf{key}_j^b, M_j^b)$.
   - It outputs $x_b = 0$ if at least one $\boldsymbol{u}_j = 0^\kappa$, and outputs 1 otherwise.

Figure 20: $(2,0)$-LTOT Protocol

Hybrid Real**.** This hybrid is the real-life execution.

Hybrid Hyb_1**.** In this hybrid, we let the simulator $\mathcal{S}$ which accesses both the inputs (x_0, x_1) of S and the input b of R to emulate the protocol. $\mathcal{S}$ never follows the protocol to erase any key_j^i from its local view.

Apparently, Real and Hyb_1 are identical from the adversary's perspective.

Hybrid Hyb_2**.** In this hybrid, for each $i \in \{0,1\}$, the simulator $\mathcal{S}$ samples two pairs of random subsets (I_0^i, J_0^i) and (I_1^i, J_1^i) (every subset has a size of κ), conditioned on $(I_0^i \cup J_0^i) \cap (I_1^i \cup J_1^i) = \emptyset$. For more details, $\mathcal{S}$ first randomly picks subsets (I_0^i, J_0^i), then samples (I_1^i, J_1^i) from $[4\kappa]/(I_0^i \cup J_0^i)$ conditioned on $|I_0^i \cap J_0^i| = |I_1^i \cap J_1^i|$. The subsets (I, J_0, J_1) sampled by R and S are replaced by $(I_{x_b}^b, J_{x_0}^0, J_{x_1}^1)$.

For each $i \in \{0,1\}$, $\mathcal{S}$ modifies the sampling process of LROTP encryption to be independent of the value of x_i:

- For $c \in \{0,1\}$ and each $j \in J_c^i$, encrypt the bit c as $M_j^i \leftarrow \mathsf{Enc}(\mathsf{key}_j^i, c^\kappa)$.
- For the remaining coordinates $j \notin J_0^i \cup J_1^i$, output a random ciphertext $M_j^i \leftarrow \mathsf{oEnc}$.

We leverage the security of the underlying LROTP scheme to argue the indistinguishability between Hyb_1 and Hyb_2. We only exhibit the proof for the case of $i = b$ (the proof of $i = 1 - b$ is analogous). For $i = b$, the only difference we made in Hyb_2 is to fix every random ciphertext $M_j^i \leftarrow \mathsf{oEnc}$ $(j \in J_{1-x_b}^b)$ by an encryption of $1-x_b$. Fix any $j \in J_{1-x_b}^b$; we can reduce the modification of M_j^i to the security game of distinguishing $\mathsf{LROTP}_{(1-x_b)^\kappa}$ from $\mathsf{LROTP}_{\mathsf{rnd}}$. Concretely, we fix the

randomness of $\mathcal{S}$ except $(\mathsf{key}_j^b, M_j^b)$. Given an instance (key, M) of the LROTP security game, we can emulate the views of S and R before the erasure using only $\mathsf{key}_j^b = \mathsf{key}$ and emulate the views after erasure using only $M_j^b = M$, the security of M_j^i then follows from Lemma 3.21. Applying a standard hybrid argument on the subset $J_{1-x_b}^b$ suffices to prove the indistinguishability between Hyb_1 and Hyb_2.

Hybrid Hyb_3. In this hybrid, $\mathcal{S}$ discards the instances of $I_0^i \cap J_0^i = \emptyset$ for some $i \in \{0,1\}$, and manipulates internal random bits of oEnc to ensure the invocations of oEnc in Hyb_2 never produce decodable ciphertexts M_j^i (i.e., $(\mathsf{key}_j^i)^T M_j^i \notin \{0^\kappa, 1^\kappa\}$). In this way, we can guarantee the absolute correctness of decryption.

Since the event $I_0^i \cap J_0^i = \emptyset$ only occurs with probability $\binom{3k}{k}/\binom{4k}{k} = \mathsf{negl}(\kappa)$, while the event $(\mathsf{key}_j^i)^T M_j^i \in \{0^\kappa, 1^\kappa\}$ only occurs with probability $O(\kappa 2^{-\kappa}) = \mathsf{negl}(\kappa)$ in Hyb_2, we have that Hyb_2 and Hyb_3 are statistically close.

Hybrid Ideal. This hybrid is the ideal world simulator. The simulator $\mathcal{S}$ works as follows:

- $\mathcal{S}$ generates the subsets (I_0^i, J_0^i) and (I_1^i, J_1^i) $(i \in \{0,1\})$ as discussed in Hyb_2, except for additionally conditioned on $I_0^0 \cap J_0^0 \neq \emptyset$ and $I_0^1 \cap J_0^1 \neq \emptyset$.
- For each $i \in \{0,1\}$, $\mathcal{S}$ prepares $[4\kappa]$ pairs of a secret key and a ciphertext $(\mathsf{key}_j^i, M_j^i)$: if $j \in J_c^i$ for some $c \in \{0,1\}$, M_j^i encrypts the string c^κ; otherwise M_j^i simply encrypts a random element which is not coincided with $\{0^\kappa, 1^\kappa\}$.
- The state translation functions T_S and T_R emulate the execution of $\mathsf{ErasureOT}$ according to the sampled values, where S uses $(J_0, J_1) = (J_{x_0}^0, J_{x_1}^1)$ and R uses $I = I_{x_b}^b$.

  It is clear that $\mathsf{Hyb}_3 \equiv \mathsf{Ideal}$.

Unconditionally Secure $(2,0)$-LTMPC. Substituting the $\mathsf{ErasureOT}$ protocol into the transformation from LTOT to 2-party LTMPC (Section 5.1), we immediately obtain a $(2,0)$-LTMPC protocol assuming secure erasures of honest parties.

Lemma 7.2. *Let $\ell = \ell(\kappa)$ be a leakage bound. Assuming honest parties can erase their local views, every 2-party functionality $\mathcal{F}$ admits an unconditionally secure $(2,0,\ell)$-LTMPC implementation.*

7.3 Unconditionally Secure Honest-Dealer LTMPC

In this section, we propose a protocol that securely implements a dealer-respect LTMPC with an honest dealer P_i, assuming an ideal functionality of BDOZ triples preprocessing. To align with the notation of our BCLSS scheme, the syntax of the BDOZ MAC used here is slightly different from the one introduced in the overview. Specifically, the message x is an element of $\mathbb{F}_2$, while the key $K = (\Delta, \zeta)$ and $\mathsf{MAC}_K(x)$ are over an extension field $\mathbb{F} = \mathbb{F}_{2^k}$ for some $k = \ell + \omega(\log \kappa)$. It is not hard to verify that the security is not compromised after this mild modification.

Notation. Let Δ_i be the global secret key of dealer P_i. A BDOZ MAC of a message $x \in \mathbb{F}_2$ under key $K = (\Delta_i, \zeta) \in \mathbb{F}^2$ is defined as $\mathsf{MAC}_K(x) = (x, \zeta + \Delta_i x) \in \mathbb{F}_2 \times \mathbb{F}$. Given an additive sharing $\langle \alpha \rangle_n = (\alpha_1, \ldots, \alpha_n)$, the BDOZ authenticated sharing $[\alpha]_n$ under secret keys $\{K_j = (\Delta_i, \zeta_j)\}_{j \neq i}$ is

$$[\alpha]_n^i = ([\alpha]_{n,i}^i := \{\alpha_i, (K_j)_{j \neq i}\}, [\alpha]_{n,j}^i := \{\mathsf{MAC}_{K_j}(\alpha_j)\}_{j \neq i}).$$

The BDOZ authenticated n-BCLSS $[\alpha]_n^{\mathsf{BCL}}$ can be defined in an analogous way from an n-BCLSS $\langle\alpha\rangle_n^{\mathsf{BCL}} = (\alpha_1, \ldots, \alpha_n)$. The BDOZ authenticated triple of $(a, b, c = ab)$ is just $([a]_n^i, [b]_n^i, [c]_n^i)$.

For two BDOZ MACs $\mathsf{MAC}_{K_1}(x_1)$ and $\mathsf{MAC}_{K_2}(x_2)$ under keys $K_1 = (\Delta_i, \zeta_1)$ and $K_2 = (\Delta_i, \zeta_2)$, we henceforth denote $(\Delta_i, \zeta_1+\zeta_2)$ as K_1+K_2. Then $\mathsf{MAC}_{K_1}(x_1)+\mathsf{MAC}_{K_2}(x_2) = \mathsf{MAC}_{K_1+K_2}(x_1+x_2)$. In this way, we can homomorphically compute

$$[\alpha]_n^i + [\beta]_n^i := (\{\alpha_i + \beta_i, (K_{\alpha,j} + K_{\beta,j})_{j\neq i}\}, \{\mathsf{MAC}_{K_{\alpha,j}}(\alpha_j) + \mathsf{MAC}_{K_{\beta,j}}(\beta_j)\}_{j\neq i}),$$

where the two BDOZ sharings $[\alpha]_n$ and $[\beta]_n$ are under keys $\{K_{\alpha,j} = (\Delta_i, \zeta_{\alpha,j})\}_{j\neq i}$ and $\{K_{\beta,j} = (\Delta_i, \zeta_{\beta,j})\}_{j\neq i}$, respectively. Moreover, to compute $[\alpha]_n^i + c$ for some constant $c \in \mathbb{F}_2$, we simply let the dealer P_i add c to its share.

BDOZ Multiplication Gate. To start with, we first consider the problem of instantiating multiplication gates. We define an n-party functionality $\mathcal{F}_{\mathsf{Mult}}^{\mathsf{BDOZ},i}$ of multiplying two BDOZ sharings and provide an n-LTMPC protocol $\pi_{\mathsf{Mult}}^{\mathsf{BDOZ},i}$ to compute it in the $\mathcal{F}_{\mathsf{Prep}}^i$-hybrid model which is an ideal functionality that outputs random BDOZ triples.

Functionality $\mathcal{F}_{\mathsf{Mult}}^{\mathsf{BDOZ},i}$

Running with parties $P_1, \ldots, P_n$ and a simulator $\mathcal{S}$ where the party P_i is designated as dealer, $\mathcal{F}_{\mathsf{Mult}}^{\mathsf{BDOZ},i}$ works as follows:

- Upon receiving input $(\mathsf{mul}-\mathsf{input}, P_i, [\alpha]_{n,i}^i, [\beta]_{n,i}^i, \Delta_i)$ from dealer P_i and $(\mathsf{mul}-\mathsf{input}, P_j, [\alpha]_{n,j}^i, [\beta]_{n,j}^i)$ from other parties P_j, notify $\mathcal{S}$ about the computation.
- Verify the MACs of $[\alpha]_{n,j}^i$ and $[\beta]_{n,j}^i$ for all $j \neq i$.
- Decode the additive sharings $\langle\alpha\rangle_n$ and $\langle\beta\rangle_n$. Then randomly sample $(\langle\gamma\rangle_{n,1}, \ldots, \langle\gamma\rangle_{n,n}) \in \mathbb{F}_2^n$ conditioned on

$$\sum_{i=1}^n \langle\gamma\rangle_{n,i} = \left(\sum_{i=1}^n \langle\alpha\rangle_{n,i}\right) \cdot \left(\sum_{i=1}^n \langle\beta\rangle_{n,i}\right).$$

- Generate a fresh authenticated sharing $[\gamma]_n^i$ from $\langle\gamma\rangle_n$ under the same global secret key Δ_i. Once $\mathcal{S}$ acknowledges, distribute the share of $[\gamma]_n^i$ to every party.

Figure 21: The BDOZ Multiplication Functionality

Assuming the dealer P_i is honest, we construct a simulator $\mathcal{S}$ for $\pi_{\mathsf{Mult}}^{\mathsf{BDOZ},i}$. We additionally allow $\mathcal{S}$ to interact the global secret key Δ_i, but only uses it to produce state translation functions of honest parties, thus does not compromise the security.

1. $\mathcal{S}$ randomly samples $\epsilon, \delta \xleftarrow{\$} \mathbb{F}_2$ and random BDOZ sharings $[\epsilon]_n^i, [\delta]_n^i$. We describe in detail how these sharings are generated: $\mathcal{S}$ first randomly generates $\langle\epsilon\rangle_n, \langle\delta\rangle_n$, then it signs the shares of every honest party $P_j \neq P_i$ with fresh keys $K_{\epsilon,j}, K_{\delta,j}$; for a corrupted party P_j, $\mathcal{S}$ simply prepares random $w_{\epsilon,j}, w_{\delta,j} \xleftarrow{\$} \mathbb{F}$ and assigns $[\epsilon]_{n,j}^i = (\langle\epsilon\rangle_{n,j}, w_{\epsilon,j}), [\delta]_{n,j}^i = (\langle\delta\rangle_{n,j}, w_{\delta,j})$. The simulation of corrupted parties is completely independent of the secret key Δ_i, thus the security of BDOZ MACs is maintained. In the state translation function of dealer P_i, $\mathcal{S}$ generates $\zeta_{\epsilon,j}, \zeta_{\delta,j}$ via $w_{\epsilon,j} - \Delta_i \cdot \langle\epsilon\rangle_{n,j}$ and $w_{\delta,j} - \Delta_i \cdot \langle\delta\rangle_{n,j}$ respectively.

Protocol $\pi^{\mathsf{BDOZ},i}_{\mathsf{Mult}}$

$P_1, \ldots, P_n$ evaluate the n-party function $\mathcal{F}^{\mathsf{BDOZ},i}_{\mathsf{Mult}}$ where the party P_i is designated as dealer and holds the global secret key Δ_i:

- The parties call $\mathcal{F}^i_{\mathsf{Prep}}$ to prepare a random BDOZ triple $([a]^i_n, [b]^i_n, [c]^i_n)$.
- Each party $P_j \neq P_i$ opens its share of $[\epsilon]^i_n = [\alpha]^i_n - [a]^i_n$ and $[\delta]^i_n = [\beta]^i_n - [b]^i_n$ to dealer P_i. P_i verifies the validity of $[\epsilon]^i_n$ and $[\delta]^i_n$. If any verification fails, it notifies every party to abort; otherwise, it gets ϵ and δ.
- P_i sends ϵ and δ to other parties (using point-to-point channels). Then each party locally computes its share of
$$[\gamma]^i_n = \epsilon\delta + \epsilon[b]^i_n + [a]^i_n\delta + [c]^i_n.$$

Figure 22: BDOZ Multiplication Protocol

Functionality $\mathcal{F}^i_{\mathsf{Prep}}$

Running with parties $P_1, \ldots, P_n$ and a simulator $\mathcal{S}$ where the party P_i is designated as dealer, $\mathcal{F}^i_{\mathsf{Prep}}$ works as follows:

Randomness Generation:

- Receive the global secret key Δ_i from dealer P_i and notify $\mathcal{S}$ about the computation.
- Sample a random n-party BDOZ share $[r]^i_n$ under Δ_i.
- Once $\mathcal{S}$ acknowledges, distribute the corresponding share of $[r]^i_n$ to each party.

Triple Generation:

- Receive the global secret key Δ_i from dealer P_i and notify $\mathcal{S}$ about the computation.
- Sample a random n-party BDOZ triple $([a]^i_n, [b]^i_n, [c]^i_n = [ab]^i_n)$ under Δ_i. Notify to $\mathcal{S}$ about the computation.
- Once $\mathcal{S}$ acknowledges, distribute the corresponding share of $([a]^i_n, [b]^i_n, [c]^i_n)$ to each party.

Figure 23: The BDOZ Preprocessing Functionality

2. For each corrupted party P_j, $\mathcal{S}$ computes P_j's *ideal* share of $[a]_n^i = [\epsilon]_n^i + [\alpha]_n^i, [b]_n^i = [\delta]_n^i + [\beta]_n^i$ as well as
$$[c]_n^i = [\gamma]_n^i - \epsilon\delta - \epsilon[b]_n^i - [a]_n^i\delta$$
based on P_j's ideal state. $\mathcal{S}$ then sends the corrupted parties' shares of $([a]_n^i, [b]_n^i, [c]_n^i)$ to the environment and receives shares of $([\epsilon]_n^i, [\delta]_n^i)$. It aborts if any such share is distinct from the recorded values.

3. The state translation function T_j of an honest party P_j can be obliviously emulated using P_j's ideal state and the messages from corrupted parties.

Note that the simulation process honestly emulates the real-life distribution unless in Step 2 the environment successfully forges $\mathsf{MAC}_{K_{\alpha,j}}(\alpha'_j)$ for some $\alpha'_j \neq \langle\alpha\rangle_{n,j}$ or $\mathsf{MAC}_{K_{\beta,j}}(\beta'_j)$ for some $\beta'_j \neq \langle\beta\rangle_{n,j}$, which happens with probability at most $2^\ell/|\mathbb{F}| = \mathsf{negl}(\kappa)$ by our parameter choice. Therefore, we conclude that the real world and the ideal world are statistically close.

Dealer-Respect LTMPC. We then enhance the $\mathcal{F}_{\mathsf{Mult}}$-LTMPC (Theorem 4.1) to achieve full security, by replacing the additive sharings used in π_{MLTMPC} (Figure 6) with BDOZ authenticated sharings and replacing oracle calls to $\mathcal{F}_{\mathsf{Mult}}$ with $\pi_{\mathsf{Mult}}^{\mathsf{BDOZ},i}$ accordingly. This approach yields a maliciously secure dealer-respect LTMPC π_{DLTMPC}^i in the $\mathcal{F}_{\mathsf{Prep}}^i$-hybrid model:

Lemma 7.3. *Let $\ell = \ell(\kappa)$ be a leakage bound. For any $n \geq 2$ and dealer P_i, every n-party functionality $\mathcal{F}$ admits a dealer-respect protocol π_{DLTMPC}^i in the $\mathcal{F}_{\mathsf{Prep}}^i$-hybrid model. Furthermore, the protocol π_{DLTMPC}^i is an unconditionally secure malicious $(n, n-2, \ell)$-LTMPC with abort implementation of $\mathcal{F}$ if the dealer P_i is honest.*

Proof Sketch. We intend to adopt the proof strategy of Theorem 4.1 (except that the simulator will verify that the corrupted parties indeed provide valid BDOZ MACs), whereas we have to resolve several issues:

1. In the input sharing and output reconstruction phases of π_{DLTMPC}^i, every party except the owner P_j sends its share of $\boldsymbol{x}_j$ (resp. $\boldsymbol{y}_j$) to the dealer P_i. Although P_i is assumed to be honest, the adversary can obtain information about those shares by launching a BCL leakage attack on P_i. This, however, does not pose any problem, because in the proof of Theorem 4.1 we implicitly prove that the security even holds when $\langle\boldsymbol{x}_j\rangle_{n,[n]/j}$ and $\langle\boldsymbol{y}_j\rangle_{n,[n]/j}$ can be revealed to the adversary in their entirety.

2. We need to show how to simulate the MACs. To this end, we let the simulator know the global secret key Δ_i, but it is used only for simulating honest parties' view. Such a relaxation does not break the security of BDOZ MACs.

   In this way, we can sample the secret key K_j for an honest party $P_j \neq P_i$, and determine the $\mathsf{MAC}_{K_j}(\langle a\rangle_{n,j})$ as long as we obliviously simulate the additive sharing $\langle a\rangle_{n,j}$. Moreover, we sample a random $w_j \xleftarrow{\$} \mathbb{F}$ and treat $\mathsf{MAC}_{K_j}(\langle a\rangle_{n,j})$ as $(\langle a\rangle_{n,j}, w_j)$ for a corrupted P_j. We can recover the secret key $K_j = (\Delta_i, \zeta_j)$ in the state translation function T_i as $\zeta_j = w_j - \Delta_i \cdot \langle a\rangle_{n,j}$.

□

Protocol π^i_{DLTMPC}

Parties $P_1, \ldots, P_n$ evaluate an n-party function $\mathcal{F}$ where P_i is designated as dealer and holds the global secret key Δ_i.

- Construct a strong parity-LTC C_{pr} of the deterministic function $\boldsymbol{y} \leftarrow \mathcal{F}'(\boldsymbol{x}, \boldsymbol{r}_{\mathcal{F}'})$ defined in Figure 6.
- $P_1, \ldots, P_n$ computes a BDOZ sharing of each wire of C_{pr} in topological order as follows:
  1. For each input wire x, the parties first pick a random $[a]^i_n \leftarrow \mathcal{F}^i_{\mathsf{Prep}}$:
     - If x carries an input bit of dealer P_i, each party $P_j \neq P_i$ convinces P_i by opening $\mathsf{MAC}_{K_{a,j}}(\langle a \rangle_{n,j})$. Then P_i gets a and broadcasts $\delta = x - a$.
     - If x carries an input bit of some $P_j \neq P_i$, each $P_k \notin \{P_i, P_j\}$ opens $\mathsf{MAC}_{K_{a,k}}(\langle a \rangle_{n,k})$. P_i verifies the correctness of MACs and sends $\{\langle a \rangle_{n,k}\}_{k \neq j}$ to P_j. In this way, P_j gets a and broadcasts $\delta = x - a$.

     The parties then use $[x]^i_n = \delta + [a]^i_n$.
  2. The parties call $\mathcal{F}^i_{\mathsf{Prep}}$ $|\boldsymbol{r}_{\mathcal{F}'}|$-times to produce $[\boldsymbol{r}_{\mathcal{F}'}]^i_n$ of a random auxiliary input $\boldsymbol{r}_{\mathcal{F}'}$.
  3. For each random wire r of C_{pr}, the parties call $\mathcal{F}^i_{\mathsf{Prep}}$ to produce $[r]^i_n$.
  4. For each addition gate $g \in C_{\mathsf{pr}}$ with input wires α, β and output wire γ, each party locally adds its shares of $[\alpha]^i_n$ and $[\beta]^i_n$ to obtain $[\gamma]^i_n$.
  5. For each multiplication gate $g \in C_{\mathsf{pr}}$ with input wires α, β and output wire γ, the parties invoke $\pi^{\mathsf{BDOZ},i}_{\mathsf{Mult}}$ to compute $[\gamma]^i_n$.
  6. For each output wire y:
     - If y carries an output bit of dealer P_i, each party $P_j \neq P_i$ opens its share $\mathsf{MAC}_{K_{y,j}}(\langle y \rangle_{n,j})$ to P_i. P_i verifies the correctness of the MACs and reconstructs y.
     - If y carries an output bit of some $P_j \neq P_i$, then each party $P_k \notin \{P_i, P_j\}$ opens its share $\mathsf{MAC}_{K_{y,k}}(\langle y \rangle_{n,k})$ to P_i. P_i checks the consistency of MACs, then sends $(\langle y \rangle_{n,k})_{k \neq j}$ to P_j so that P_j can reconstruct y.

Figure 24: Dealer-Respect LTMPC Protocol

7.4 Beaver Triples Preprocessing

In this section, we introduce a protocol π^i_{Prep} which implements the functionality $\mathcal{F}^i_{\mathsf{Prep}}$. Since we can keep only the first coordinate of a random BDOZ triple $([a]^i_n, [b]^i_n, [c]^i_n)$ to obtain a random BDOZ sharing, we only discuss how to generate triples. For this, we first formally define the notion of 2-source extractors for BDOZ triples:

Definition 7.4 (Two-Source Extractor for BDOZ Triples)**.** Let $\mathsf{Ext} : \mathbb{F}_2^N \times \mathbb{F}_2^D \to \mathbb{F}_2^3$ and $\mathsf{Rand} : \mathbb{F}_2^M \to \mathbb{F}_2^N$ be two deterministic functions. We say that $(\mathsf{Ext}, \mathsf{Rand})$ is a $(K_1, K_2, K_3, \epsilon)$-2-source extractor for Beaver triples if it satisfies the following conditions:

- *Linearity:* For every $X_1, X_2 \in \mathbb{F}_2^N$ and every $Z \in \mathbb{F}_2^D$,

$$\mathsf{Ext}(X_1, Z) + \mathsf{Ext}(X_2, Z) = \mathsf{Ext}(X_1 + X_2, Z).$$

- *Randomness Extraction:* Ignoring the Rand function, Ext itself is a 2-source extractor. Namely, for every pair of independent random variables X, Z such that $\mathsf{H}_\infty(X) \geq K_1, \mathsf{H}_\infty(Z) \geq K_3$,

$$\Delta(\mathsf{Ext}(X, Z), U_3) \leq \epsilon.$$

- *Beaver Triples Extraction:* For every $Y \in \mathbb{F}_2^M$ and every $Z \in \mathbb{F}_2^D$, $\mathsf{Ext}(\mathsf{Rand}(Y), Z)$ outputs a valid Beaver triple $(u, v, u \cdot v)$. Note that the linearity and the correctness of Beaver triples extraction ensure that for any length-n additive secret sharing $\langle \mathsf{Rand}(Y) \rangle_n$, $\sum_{i=1}^n \mathsf{Ext}(\langle \mathsf{Rand}(Y) \rangle_{n,i}, Z)$ is exactly the Beaver triple $\mathsf{Ext}(\mathsf{Rand}(Y), Z)$.

  Moreover, the function $\mathsf{Ext}(\mathsf{Rand}, \cdot)$ extracts random Beaver triples. Let U_{Beaver} be the uniform distribution over $\{(u, v, u \cdot v) : u, v \in \mathbb{F}_2\}$. Then for every pair of independent random variables Y, Z such that $\mathsf{H}_\infty(Y) \geq K_2, \mathsf{H}_\infty(Z) \geq K_3$, we require that

$$\Delta(\mathsf{Ext}(\mathsf{Rand}(Y), Z), U_{\mathsf{Beaver}}) \leq \epsilon.$$

Let $\Delta \in \mathbb{F}$ be a global secret key and henceforth denote $\mathbb{F}' = \mathbb{F}_2 \times \mathbb{F}$. We can augment a 2-source-extractor for Beaver triples to an extractor for BDOZ triples, by additionally taking the functionalities $\mathsf{Sign}_\Delta : \mathbb{F}_2^N \times \mathbb{F}^N \to \mathbb{F}'^N, \mathsf{MACExt} : \mathbb{F}'^N \times \mathbb{F}_2^D \to \mathbb{F}'^3$ and $\mathsf{KeyExt}_\Delta : \mathbb{F}^N \times \mathbb{F}_2^D \to (\mathbb{F}^2)^3$ that homomorphically compute the MAC of the BDOZ extractor:

- On every input $X \in \mathbb{F}_2^N$ and $KS \in \mathbb{F}^N$, the signing algorithm $\mathsf{Sign}_\Delta(X, KS)$ outputs $(X, W) \in \mathbb{F}'^N$. The Sign_Δ function is a linear map:

$$\mathsf{Sign}_\Delta(X_1, KS_1) + \mathsf{Sign}_\Delta(X_2, KS_2) = \mathsf{Sign}_\Delta(X_1 + X_2, KS_1 + KS_2).$$

  Moreover, for any fixed $X \in \mathbb{F}_2^N$, the function $(X, W) = \mathsf{Sign}_\Delta(X, KS)$ is a bijection between (W, KS) which can be efficiently inverted.

- On every input $KS \in \mathbb{F}^N$ and $Z \in \mathbb{F}_2^D$, the key extraction algorithm $\mathsf{KeyExt}_\Delta(KS, Z)$ outputs three *uniformly random* MAC keys (K_a, K_b, K_c) conditioned on the global secret key Δ. Moreover, KeyExt_Δ admits an inverse algorithm: given any instance of MAC keys $(K_a, K_b, K_c) \in (\mathbb{F}^2)^3$, with overwhelming probability of random choice $Z \xleftarrow{\$} U_D$ we can efficiently reverse-sample a KS such that $\mathsf{KeyExt}_\Delta(KS, Z) = (K_a, K_b, K_c)$.

- The MACExt function outputs the MACs of $(a,b,c) \leftarrow \mathsf{Ext}(X,Z)$. Formally, for any $X \in \mathbb{F}_2^N, KS \in \mathbb{F}^N$ and $Z \in \mathbb{F}_2^D$, we require that

$$(M(a) := \mathsf{MAC}_{K_a}(a), M(b) := \mathsf{MAC}_{K_b}(b), M(c) := \mathsf{MAC}_{K_c}(c)) = \mathsf{MACExt}((X,W),Z)$$

holds, where $(X,W) \leftarrow \mathsf{Sign}_\Delta(X,KS)$, $(a,b,c) \leftarrow \mathsf{Ext}(X,Z)$ and $(K_a,K_b,K_c) \leftarrow \mathsf{KeyExt}_\Delta(KS,Z)$. The algorithm MACExt also admits an inverse algorithm: given any MACs $(M(a),M(b),M(c))$, with overwhelming probability of random choice $Z \xleftarrow{\$} U_D$ we can efficiently reverse-sample (X,W) such that $\mathsf{MACExt}((X,W),Z) = (M(a),M(b),M(c))$.

We provide a candidate construction of a 2-source extractor for BDOZ triples in Appendix B with an arbitrarily large entropy gap. Specifically, for any parameters $\lambda, \epsilon > 0$, there exists a (K_1,K_2,K_3,ϵ)-2-source extractor for BDOZ triples where $\min\{N-K_1, M-K_2, D-K_3\} \geq \lambda$ and $\max\{N,M,D\} = \mathsf{poly}(\lambda, \log(1/\epsilon))$.

Security. Suppose that there are at least two honest parties. By appropriately relabeling the parties, we may assume that the dealer is P_1 and the honest parties are $P_2,\ldots,P_k$ without loss of generality. We use a hybrid argument to prove that π^i_{Prep} securely computes the functionality $\mathcal{F}^i_{\mathsf{Prep}}$.

Hybrid Real**.** This hybrid is the real-life execution.

Hybrid Hyb_1**.** In this hybrid, we invoke the simulator S_{Ext} to emulate π_j for every honest party P_j ($j \in [2,k]$). Concretely, we first compute $(a_j,b_j,c_j) \leftarrow \mathsf{Ext}(X_j^1 + X_j^2, Z)$ and $(X_j^t, W_j^t) \leftarrow \mathsf{Sign}_{\Delta_1}(X_j^t, KS_j^t)$ ($t \in \{1,2\}$), then the view of P_j in Line 5 is simulated by $T_j((X_j^2,W_j^2),(M(a_j),M(b_j),M(c_j)))$ and the view of P_1 is simulated by $T_1((X_j^1,W_j^1),Z)$.

By the security of the underlying $(2,0,\ell)$-LTMPC, Real and Hyb_1 are statistically indistinguishable under ℓ bits of leakage.

Hybrid Hyb_2**.** In this hybrid, we generate (a_k,b_k,c_k) of the last honest party P_k in the following way: uniformly sample $(a,b) \xleftarrow{\$} U_2$ and set $(a_k,b_k,c_k) = (a,b,ab) - \sum_{j\neq k}(a_j,b_j,c_j)$. The simulation for π_k is changed accordingly using S_{Ext}.

Lemma 7.5. $\mathsf{Hyb}_1 \approx_S \mathsf{Hyb}_2$ *under ℓ bits of leakage.*

Proof. We first consider a game of attacking the Beaver triples extraction property of (Y,Z), defined as follows:

- The game chooses $r \xleftarrow{\$} U_1$ and samples $(Y,Z) \xleftarrow{\$} (U_M,U_D)$. If $r=0$, it computes $(a,b,ab) \leftarrow \mathsf{Ext}(\mathsf{Rand}(Y),Z)$; if $r=1$ it merely samples $(a,b,ab) \xleftarrow{\$} U_{\mathsf{Beaver}}$.
- The adversary $\mathcal{A}_{\mathsf{Beaver}}$ receives (a,b,ab), then it launches an $\mathcal{L}^{2,0,\ell+6n(1+\log|\mathbb{F}|)}_{\mathsf{BCL}}$-leakage attack on (Y,Z).
- $\mathcal{A}_{\mathsf{Beaver}}$ outputs a guess $r' \in \{0,1\}$. It wins the game if and only if $r'=r$.

We show that if Hyb_1 and Hyb_2 are δ-distinguishable by adversary $\mathcal{A}_{\mathsf{Prep}}$, then $\mathcal{A}_{\mathsf{Beaver}}$ can also win the above game with advantage at least $\delta - 2^{-\kappa}$. To achieve this, we reconstruct the view of $\mathcal{A}_{\mathsf{Prep}}$ by launching an $\mathcal{L}^{2,0,\ell+6n(1+\log|\mathbb{F}|)}_{\mathsf{BCL}}$ leakage attack on (Y,Z):

Protocol π^i_{Prep}

$P_1, \ldots, P_n$ evaluate the n-party function $\mathcal{F}^i_{\mathsf{Prep}}$ where the party P_i is designated as dealer and holds the global secret key Δ_i:

1. Let $(\mathsf{Ext}, \mathsf{Rand}, \mathsf{Sign}_{\Delta_i}, \mathsf{MACExt}, \mathsf{KeyExt}_{\Delta_i})$ be a $(K_1, K_2, K_3, \epsilon = \mathsf{negl}(\kappa))$-2-source extractor for BDOZ triples satisfying

$$\min\{N - K_1, M - K_2, D - K_3\} \geq \ell + 6n(1 + \log|\mathbb{F}|) + \kappa.$$

   P_i samples $Y \xleftarrow{\$} U_M, X \leftarrow \mathsf{Rand}(Y)$ and a random additive secret sharing $\langle X \rangle_n = (X_1, \ldots, X_n)$.

2. For each $j \neq i$, P_i randomly samples a $KS_j \xleftarrow{\$} \mathbb{F}^N$ as well as 2-party additive sharings $\langle X_j \rangle_2 = (X_j^1, X_j^2)$ and $\langle KS_j \rangle_2 = (KS_j^1, KS_j^2)$. After that, it sends $(X_j^2, W_j^2) \leftarrow \mathsf{Sign}_{\Delta_i}(X_j^2, KS_j^2)$ to P_j.

3. P_i clears all records pertaining to shares of X, except for its share X_i and $\{X_j^1\}_{j \neq i}$. In addition, it also erases Y and the random bits used in Rand.

4. P_i randomly samples $Z \xleftarrow{\$} U_D$ and computes $(a_i, b_i, c_i) \leftarrow \mathsf{Ext}(X_i, Z)$.

5. For every $j \neq i$, P_i computes the MAC keys $(K_{a,j}, K_{b,j}, K_{c,j}) \xleftarrow{\$} \mathsf{KeyExt}_{\Delta_i}(KS_j, Z)$. Then P_j and P_i run a $(2, 0, \ell)$-LTMPC π_j (see Lemma 7.2) with the following syntax: on input (X_j^2, W_j^2) of P_j and $(Z, (X_j^1, W_j^1) \leftarrow \mathsf{Sign}_{\Delta_i}(X_j^1, KS_j^1))$ of P_i, π_j outputs

$$(M(a_j), M(b_j), M(c_j)) \leftarrow \mathsf{MACExt}((X_j^1 + X_j^2, W_j^1 + W_j^2), Z)$$

   to P_j. Here, P_j is the first party of π_j who acts as the sender of the underlying LTOT and P_i is the second party who acts as the receiver.

6. The dealer P_i outputs $[a]^i_{n,i} = (a_i, \{K_{a,j}\}_{j \neq i}), [b]^i_{n,i} = (b_i, \{K_{b,j}\}_{j \neq i})$ and $[c]^i_{n,i} = (c_i, \{K_{c,j}\}_{j \neq i})$. Each remaining party $P_j \neq P_i$ outputs $[a]^i_{n,j} = M(a_j), [b]^i_{n,j} = M(b_j), [c]^i_{n,j} = M(c_j)$.

Figure 25: BDOZ Triples Preprocessing Protocol

- $\mathcal{A}_{\mathsf{Beaver}}$ receives the challenge (a, b, ab) and generates a random global secret key Δ_1.
- $\mathcal{A}_{\mathsf{Beaver}}$ samples the random shares $\{X_j\}_{j\neq k} \cup \{\langle X_j\rangle_2\}_{j\neq 1,k}$ as well as X_k^1. Then X_k can be expressed as $\mathsf{Rand}(Y) - \sum_{j\neq k} X_j$ and X_k^2 equals $X_k - X_k^1$. Both of them are considered as functions of Y.
- $\mathcal{A}_{\mathsf{Beaver}}$ samples random sharings $\{\langle KS_j\rangle_2\}_{j\neq 1}$ and computes each W_j^t from $\mathsf{Sign}_{\Delta_1}(X_j^t, KS_j^t)$. Then it launches a $3n\log|\mathbb{F}|$-bit BCL leakage attack on Z to recover $(K_{a,j}, K_{b,j}, K_{c,j})_{j\neq 1} \leftarrow \mathsf{KeyExt}_{\Delta_1}(KS_j, Z)$.
- $\mathcal{A}_{\mathsf{Beaver}}$ launches another $3n$-bit BCL leakage attack on Z to determine $(a_j, b_j, c_j) \leftarrow \mathsf{Ext}(X_j, Z)$ for every $P_j \neq P_k$. Then the shares (a_k, b_k, c_k) of P_k can be computed as $(a_k, b_k, c_k) = (a, b, ab) - \sum_{j\neq k}(a_j, b_j, c_j)$. Each of the MACs $(M(a_j), M(b_j), M(c_j))$ is a function of (a_j, b_j, c_j) and $(K_{a,j}, K_{b,j}, K_{c,j})$, where the latter term has been computed in the last step.
- By carefully verifying the design of the underlying $(2, 0)$-LTMPC, we find that for every corrupted party P_j, only the last message (in the output reconstruction phase) of π_j might depend on P_i's input Z, which has length $|M(a_j)|+|M(b_j)|+|M(c_j)| = 3(1+\log|\mathbb{F}|)$. Therefore $\mathcal{A}_{\mathsf{Beaver}}$ can emulate the corrupted parties' views by launching a $3n(1+\log|\mathbb{F}|)$-bit BCL leakage attack on Z.
- Moreover, the view of (1) every honest party P_j ($j \in [2, k]$); (2) P_1 in Line 1,2,3 can be fully determined by Y and sampled values, hence any leakage query to them made by $\mathcal{A}_{\mathsf{Prep}}$ can be reduced to a BCL leakage attack on Y of the same length. In contrast, the view of P_1 in Line 4,5,6 depends only on Z and sampled values, as do the leakage queries of $\mathcal{A}_{\mathsf{Prep}}$ to it.

Employing a standard computational leveraging technique, $\mathcal{A}_{\mathsf{Beaver}}$ can be converted to an adversary with advantage $(\delta - 2^{-\kappa})/4$ that makes a $\mathcal{L}_{\mathsf{BCL}}^{2,0,\ell+6n(1+\log|\mathbb{F}|)}$ leakage attack *before* it receives the challenge (a, b, ab). This can be done by simply "guessing" the challenge, and outputting a random b' if it finds that the guess is wrong after receiving the true challenge. Then Lemma 7.5 then follows from the Beaver triple extraction Property since $\min\{M - K_2, D - K_3\} \geq \ell + 6n(1 + \log|\mathbb{F}|) + \kappa$. □

Hybrid Hyb_3. In this hybrid, we switch the outputs of every P_j ($j \in [2, k-1]$) to a random $(a_j, b_j, c_j) \xleftarrow{\$} U_3$, while the output of P_k is still computed as $(a_k, b_k, c_k) = (a, b, ab) - \sum_{j\neq k}(a_j, b_j, c_j)$. We use a sequence of intermediate hybrids to show the indistinguishability between Hyb_2 and Hyb_3 (indexed by $\mathsf{Hyb}_3^k, \mathsf{Hyb}_3^{k-1}, \ldots, \mathsf{Hyb}_3^2$): in Hyb_3^T, only the outputs of $P_T, P_{T+1}, \ldots, P_{k-1}$ are replaced by random values.

Clearly $\mathsf{Hyb}_3^k \equiv \mathsf{Hyb}_2$ and $\mathsf{Hyb}_3^2 \equiv \mathsf{Hyb}_3$. We then only need to show that $\mathsf{Hyb}_3^{T+1} \approx_S \mathsf{Hyb}_3^T$ for every $T \in [2, k-1]$:

Lemma 7.6. $\mathsf{Hyb}_3^{T+1} \approx_S \mathsf{Hyb}_3^T$ *under ℓ bits of leakage.*

Proof Sketch. We reduce the indistinguishability between $(\mathsf{Hyb}_3^{T+1}, \mathsf{Hyb}_3^T)$ to the randomness extraction property of (X_T, Z): distinguish $(a, b, c) \leftarrow \mathsf{Ext}(X_T, Z)$ from a random challenge $(a, b, c) \xleftarrow{\$} U_3$. We discuss how to generate the shares X_j and the Beaver triples (a_j, b_j, c_j):

- The adversary $\mathcal{A}_{\mathsf{Random}}$ randomly samples $\{X_j\}_{j\neq T,k} \cup \{\langle X_j\rangle_2\}_{j\neq 1,T,k}$ as well as X_T^1, X_k^1. To preserve the consistency of dealer P_1's view, it also samples $Y \xleftarrow{\$} U_M$ and assigns $X_k = \mathsf{Rand}(Y) - \sum_{j\neq k} X_j, X_k^2 = X_k - X_k^1$ which are functions of X_T.

- $\mathcal{A}_{\mathsf{Random}}$ applies $3n$-bit BCL leakage of Z to obtain $(a_j, b_j, c_j)_{j\in[1,T)\cup(k,n]}$ and randomly samples $(a_j, b_j, c_j) \xleftarrow{\$} U_3$ for every $j \in (T, k)$. To preserve output consistency, $\mathcal{A}_{\mathsf{Random}}$ samples $(a, b, ab) \xleftarrow{\$} U_{\mathsf{Beaver}}$ and assigns $(a_k, b_k, c_k) = (a, b, ab) - \sum_{j\neq k}(a_j, b_j, c_j)$.

  The rest of the proof is essentially the same as that of Lemma 7.5 and we omit it here. □

Hybrid Hyb_4. In this hybrid, the outputs of corrupted parties $P_{k+1}, \ldots, P_n$ are replaced by random values. For more details, we first sample a uniformly random $Z \xleftarrow{\$} U_D$. Then, on the ideal output $(M(a_j), M(b_j), M(c_j))$ of a corrupted party P_j we apply the inverse function of MACExt to find (X_j, W_j) satisfying $\mathsf{MACExt}((X_j, W_j), Z) = (M(a_j), M(b_j), M(c_j))$ and compute the additive secret sharing $\langle X_j\rangle_2$ accordingly. The next step is to generate a random secret sharing $\langle W_j\rangle_2 = (W_j^1, W_j^2)$ of W_j. By the bijection property of Sign_{Δ_1}, we can uniquely reveal KS_j^1 and KS_j^2 such that $\mathsf{Sign}_{\Delta_1}(X_j^1, KS_j^1) = (X_j^1, W_j^1)$ and $\mathsf{Sign}_{\Delta_1}(X_j^2, KS_j^2) = (X_j^2, W_j^2)$, and reveal $KS_j = KS_j^1 + KS_j^2$.

The invertibility of MACExt ensures that Hyb_3 and Hyb_4 are statistically close to each other.

Hybrid Hyb_5. In this hybrid, P_1 uses the inverse algorithm of $\mathsf{KeyExt}_{\Delta_1}$ to find a KS_j such that $\mathsf{KeyExt}_{\Delta_1}(KS_j, Z) = (K_{a,j}, K_{b,j}, K_{c,j})$ for every honest party $j \in [2, k]$. We emphasize that Hyb_4 already guarantees that $\mathsf{KeyExt}(KS_j, Z) = (K_{a_j}, K_{b,j}, K_{c,j})$ for every $j \in [k+1, n]$ because of the correctness of ideal outputs.

The indistinguishability between Hyb_4 and Hyb_5 follows from the output randomness of KeyExt.

Hybrid Ideal. This hybrid is the ideal world distribution. The simulator $\mathcal{S}$ works as follows:

- $\mathcal{S}$ randomly samples $Z \xleftarrow{\$} U_D$ and additive sharings $\langle X_2\rangle_2, \ldots, \langle X_{k-1}\rangle_2 \xleftarrow{\$} U_N^2$ along with a random share $X_k^2 \xleftarrow{\$} U_N$ (recall that there are $k \geq 2$ honest parties). Besides, utilizing the bijection property of Sign_{Δ_1}, $\mathcal{S}$ generates $W_2^2, \ldots, W_k^2 \xleftarrow{\$} \mathbb{F}^N$.
- On the ideal functionality output $(M(a_j), M(b_j), M(c_j)) \leftarrow \mathcal{F}_{\mathsf{Prep}}$ of each corrupted party P_j ($j \in [k+1, n]$), $\mathcal{S}$ uses the inverse algorithm of MACExt to find a pair (X_j, W_j) where $\mathsf{MACExt}((X_j, W_j), Z) = (M(a_j), M(b_j), M(c_j))$ and samples random sharings $\langle X_j\rangle_2, \langle W_j\rangle_2$. In this way, the simulation of corrupted parties' views is independent of Δ_1.
- To construct the state translation function T_1, $\mathcal{S}$ first uses rejection sampling to sample X_1 such that $\mathsf{Ext}(X_1, Z) = (a_1, b_1, c_1)$. Then it generates a random $Y \xleftarrow{\$} U_M$ and computes $X_k^1 = \mathsf{Rand}(Y) - X_k^2 - \sum_{j\neq k} X_j$.

  For every honest party P_j ($j \in [2, k]$), $\mathcal{S}$ inverts KS_j^2 where $\mathsf{Sign}_{\Delta_1}(X_j^2, KS_j^2) = (X_j^2, W_j^2)$. It then reversely-samples KS_j conditioned on $\mathsf{KeyExt}(KS_j, Z) = (K_{a,j}, K_{b,j}, K_{c,j})$. In this way, it also finds $KS_j^1 = KS_j - KS_j^2$ as well as $(X_j^1, W_j^1) \leftarrow \mathsf{Sign}_{\Delta_1}(X_j^1, KS_j^1)$.

  For every corrupted party P_j ($j \in [k+1, n]$), $\mathcal{S}$ recovers the unique values KS_j^1 and KS_j^2 where $\mathsf{Sign}_{\Delta_1}(X_j^t, KS_j^t) = (X_j^t, W_j^t)$ ($t \in \{1, 2\}$). Then $\mathcal{S}$ honestly emulates the execution of π_j.
- The state translation function T_j of other honest parties ($j \in [2, k]$) can be directly emulated using common knowledge (X_j^2, W_j^2) and ideal functionality output $(M(a_j), M(b_j), M(c_j)) \leftarrow \mathcal{F}_{\mathsf{Prep}}$.

Observe that the only difference between Hyb_5 and Ideal is the output of P_1: in Hyb_5 we directly compute $(a_1, b_1, c_1) \leftarrow \mathsf{Ext}(X_1, Z)$ for some $X_1 \xleftarrow{\$} U_N$, whereas in Ideal the output (a_1, b_1, c_1) is assigned with random values. It follows from the randomness extraction property of Ext that $\mathsf{Hyb}_5 \approx_\epsilon \mathsf{Ideal}$.

Combining this with Lemma 7.3, we conclude with the following lemma:

Lemma 7.7. *Let $\ell = \ell(\kappa)$ be a leakage bound. For any $n \geq 2$, every n-party functionality $\mathcal{F}$ admits a dealer-respect protocol assuming honest parties can erase their local views. Furthermore, the protocol is an unconditionally secure malicious $(n, n-2, \ell)$-LTMPC with abort implementation of $\mathcal{F}$ if the dealer P_i is honest.*

7.5 The Full Scheme

We now introduce an unconditionally secure LTMPC π_{LTMPC} that securely computes any (possibly randomized) n-party function $\mathcal{F}$ in the presence of an honest majority and secure erasure. Roughly speaking, the construction of π_{LTMPC} follows from the paradigm of designing semi-honestly secure LTMPC discussed in Section 5.3 with some necessary modifications, such as replacing the external protocol with maliciously secure (non-leakage-tolerant) MPC protocols (e.g. [RBO89]) and encoding the randomness and transcript with BDOZ n-BCLSSs. However, we need to devote more attention to the message transmission between virtual parties.

Suppose that virtual party V_i is willing to send a message m to the virtual party V_j. For this, we prepare a BDOZ n-BCLSS $[m]_n^{\mathsf{BCL},j}$ with dealer P_j. A straightforward approach is to invoke a dealer-respect LTMPC π_i with dealer P_i that outputs $[m]_n^{\mathsf{BCL},j}$, but this would lead to security issues. π_i only provides a security guarantee for an honest dealer P_i, thus the global secret key Δ_j used in computing $[m]_n^{\mathsf{BCL},j}$ might be revealed to the adversary entirely when P_i is corrupted. To resolve this issue, we only let π_i output an intermediate BDOZ n-BCLSS $[m]_n^{\mathsf{BCL},i}$, and apply a commit functionality $\mathcal{F}_{\mathsf{Com}}^{i,j}$ with dealer P_j to convert $[m]_n^{\mathsf{BCL},i}$ into $[m]_n^{\mathsf{BCL},j}$. The global secret key used for $[m]_n^{\mathsf{BCL},i}$ is fresh to protect Δ_i. For an honest party P_j, we analyze the security of this approach by considering two cases:

- P_i **is corrupted.** The adversary is permitted to optionally provide inputs which are inconsistent with $[m]_n^{\mathsf{BCL},i}$ to $\mathcal{F}_{\mathsf{Com}}^{i,j}$. But once $[m]_n^{\mathsf{BCL},j}$ is determined, the adversary is enforced to use the same shares in different invocations of V_j.
- P_i **is honest.** The adversary is enforced to use the shares of $[m]_n^{\mathsf{BCL},i}$ as inputs provided to $\mathcal{F}_{\mathsf{Com}}^{i,j}$, thus the message m is consistent between (V_i, V_j).

Security. We use a standard hybrid argument to prove the security of π_{LTMPC} in the presence of an honest majority:

Hybrid Real**.** This is the real-life distribution.

Hybrid Hyb_1**.** In this hybrid, given the inputs of all honest parties, we generate the joint view of every party in Real while emulating all invocations of the dealer-respect LTMPCs as follows:

- Sample a global secret key Δ_i for every honest party P_i.

Protocol π_{LTMPC}

$P_1, \ldots, P_n$ evaluate an n-party function $\mathcal{F}$:

- Each party P_i generates a global secret key $\Delta_i \xleftarrow{\$} \mathbb{F}$.
- Construct an unconditionally secure protocol π_{Ext} of $\mathcal{F}$ over n virtual parties $V_1, \ldots, V_n$ tolerating at most $t = \lfloor (n-1)/2 \rfloor$ malicious corruptions, where each virtual party V_i admits the same input $\boldsymbol{x}_i$ and output $\boldsymbol{y}_i$ as the actual party P_i.
- For each virtual party V_i, the parties run a dealer-respect $(n, n-2, \ell)$-LTMPC with dealer P_i to call $\mathcal{F}_{\mathsf{Int}}^i$ to prepare the internal randomness $[\boldsymbol{r}_i]_n^{\mathsf{BCL},i}$.
- $P_1, \ldots, P_n$ emulate the protocol π_{Ext}, where the transcript of each virtual party V_i is split into a BDOZ n-BCLSS $[\tau_i]_n^{\mathsf{BCL},i}$. Whenever V_i plans to send a message m to another virtual party V_j, the parties run a dealer-respect $(n, n-2, \ell)$-LTMPC with dealer P_i to call the functionality $\mathcal{F}_{\mathsf{Int}}^i$ outputting $[m]_n^{\mathsf{BCL},i}$. After that, the parties run a dealer-respect $(n, n-2, \ell)$-LTMPC with dealer P_j to call $\mathcal{F}_{\mathsf{Com}}^{i,j}$ on input $[m]_n^{\mathsf{BCL},i}$, resulting in the BDOZ n-BCLSS $[m]_n^{\mathsf{BCL},j}$ with dealer P_j. Then every party locally updates its share of $[\tau_j]_n^{\mathsf{BCL},j}$.
- For each $i \in [n]$, $P_1, \ldots, P_n$ run a dealer-respect $(n, n-2, \ell)$-LTMPC with dealer P_i to call $\mathcal{F}_{\mathsf{Int}}^i$, which decodes the output $\boldsymbol{y}_i$ for P_i.

Figure 26: n-LTMPC Protocol in Presence of Honest Majority

- Prepare a random BDOZ n-BCLSS as the internal randomness $[\boldsymbol{r}_i]_n^{\mathsf{BCL},i}$ for every honest virtual party V_i. For every corrupted virtual party V_i, we honestly emulate the function call to $\mathcal{F}_{\mathsf{Int}}^i$ to obtain the shares of honest parties.
- Sequentially emulate the execution of the protocol π_{Ext}. Specifically, for every message m delivered from V_i to V_j:
  - If P_i is a corrupted party, honestly emulate the function call to $\mathcal{F}_{\mathsf{Int}}^i$ (and record the output shares of honest parties); if P_i is honest, directly compute m from the input $\boldsymbol{x}_i$, randomness $\boldsymbol{r}_i$ as well as the transcript τ_i, and prepare a random BDOZ n-BCLSS $[m]_n^{\mathsf{BCL},i}$ (with a fresh global secret key Δ_m) as the ideal functionality output. From this, we can invoke the simulator of the underlying dealer-respect LTMPC to simulate the function call to $\mathcal{F}_{\mathsf{Int}}^i$.
  - If P_j is a corrupted party, also emulate the function call to $\mathcal{F}_{\mathsf{Com}}^{i,j}$; if P_j is honest, invoke the simulator of the underlying dealer-respect LTMPC. Once the $\mathcal{F}_{\mathsf{Com}}^{i,j}$ functionality receives the entire BDOZ BCLSS $[m]_n^{\mathsf{BCL},i}$ (especially the shares from corrupted parties), it responds to the simulator with a random BDOZ BCLSS $[m]_n^{\mathsf{BCL},j}$ of the same message m (note that we capture all outgoing messages from corrupted parties to honest parties in this way). We then update the transcript BCLSS $[\tau_j]_n^{\mathsf{BCL},j}$ accordingly.
- To decode the output $\boldsymbol{y}_i$ of a corrupted party P_i, simply emulate the oracle call to $\mathcal{F}_{\mathsf{Int}}^i$ again; when P_i is an honest party, invoke the simulator of the underlying dealer-respect LTMPC to decode $\boldsymbol{y}_i$.

By the security of the underlying dealer-respect LTMPC, we conclude that Real and Hyb_1 are statistically indistinguishable under ℓ bits of leakage.

Functionality $\mathcal{F}^i_{\mathsf{Int}}$

Running with parties $P_1, \ldots, P_n$ and a simulator $\mathcal{S}$ where the party P_i is designated as the dealer, $\mathcal{F}^i_{\mathsf{Int}}$ works as follows:

Randomness Generation:

- Upon receiving the global secret key Δ_i from dealer P_i, prepare a random BDOZ (n, ℓ)-BCLSS $[\boldsymbol{r}_i]_n^{\mathsf{BCL},i}$ and notify $\mathcal{S}$ about the computation.
- Once $\mathcal{S}$ acknowledges, send $[\boldsymbol{r}_i]_{n,j}^{\mathsf{BCL},i}$ to each party P_j.

Message Transmission:

- Upon receiving input $(\mathsf{send}, P_i, \boldsymbol{x}_i, \Delta_i, [\boldsymbol{r}_i]_{n,i}^{\mathsf{BCL},i}, [\tau_i]_{n,i}^{\mathsf{BCL},i})$ from dealer P_i and input $(\mathsf{send}, P_j, [\boldsymbol{r}_i]_{n,j}^{\mathsf{BCL},i}, [\tau_i]_{n,j}^{\mathsf{BCL},i})$ from every other party P_j, notify $\mathcal{S}$ about the computation.
- Verify the validity of $[\boldsymbol{r}_i]_n^{\mathsf{BCL},i}$ and $[\tau_i]_n^{\mathsf{BCL},i}$ under the global secret key Δ_i.
- Compute the next message m from the input $\boldsymbol{x}_i$, randomness $\boldsymbol{r}_i$ and transcript τ_i.
- Sample a random BDOZ n-BCLSS $[m]_n^{\mathsf{BCL},i}$ of message m under a fresh global key $\Delta_m \xleftarrow{\$} \mathbb{F}$. Once $\mathcal{S}$ acknowledges, distribute the corresponding share to each party.

Output Decoding:

- Upon receiving input $(\mathsf{send}, P_i, \boldsymbol{x}_i, \Delta_i, [\boldsymbol{r}_i]_{n,i}^{\mathsf{BCL},i}, [\tau_i]_{n,i}^{\mathsf{BCL},i})$ from dealer P_i and input $(\mathsf{send}, P_j, [\boldsymbol{r}_i]_{n,j}^{\mathsf{BCL},i}, [\tau_i]_{n,j}^{\mathsf{BCL},i})$ from every other party P_j, notify $\mathcal{S}$ about the computation.
- Verify the validity of $[\boldsymbol{r}_i]_n^{\mathsf{BCL},i}$ and $[\tau_i]_n^{\mathsf{BCL},i}$.
- Decode the output $\boldsymbol{y}_i$ from the input $\boldsymbol{x}_i$, randomness $\boldsymbol{r}_i$ and transcript τ_i of virtual party V_i.
- Once $\mathcal{S}$ acknowledges, send $\boldsymbol{y}_i$ to P_i.

Figure 27: The Inner Functionality

Functionality $\mathcal{F}_{\mathsf{Com}}^{i,j}$

Running with parties $P_1, \ldots, P_n$ and a simulator $\mathcal{S}$ where the party P_i is designated as the message sender and P_j is designated as the receiver, $\mathcal{F}_{\mathsf{Com}}^{i,j}$ works as follows:

- Upon receiving input $(\mathsf{commit}, P_j, \Delta_j, [m]_{n,j}^{\mathsf{BCL},i})$ from dealer P_j and input $(\mathsf{commit}, P_k, [m]_{n,k}^{\mathsf{BCL},i})$ from every other party $P_k \neq P_j$, notify $\mathcal{S}$ about the computation.
- Verify the validity of $[m]_n^{\mathsf{BCL},i}$.
- Sample a random BDOZ n-BCLSS $[m]_n^{\mathsf{BCL},j}$ of message m under Δ_j. Once $\mathcal{S}$ acknowledges, distribute the corresponding share to each party.

Figure 28: The Message Commitment Functionality

Hybrid Hyb_2. In this hybrid, we forge the ideal functionality output of $\mathcal{F}_{\mathsf{Int}}^i$ so that each honest virtual party V_i's randomness $[\boldsymbol{r}_i]_n^{\mathsf{BCL}}$ is fixed to some random encodings of all-0 strings. Moreover, for the message transmission between honest virtual parties (V_i, V_j), we also switch $[m]_n^{\mathsf{BCL},i}$ and $[m]_n^{\mathsf{BCL},j}$ to encode all-0 strings.

The indistinguishability between Hyb_1 and Hyb_2 follows from the security of the underlying BDOZ BCLSS scheme.

Hybrid Hyb_3. In this hybrid, when we generate any BDOZ BCLSS $[a]_n^{\mathsf{BCL,i}}$ with an honest dealer P_i and global secret key Δ as the ideal output of $\mathcal{F}_{\mathsf{Int}}^i$ or $\mathcal{F}_{\mathsf{Com}}^{\cdot,i}$, we delay the decision of generating the MAC key of each corrupted party P_j. Specifically, we treat $\mathsf{MAC}_{K_j}(\langle a \rangle_{n,j}^{\mathsf{BCL}})$ as $(\langle a \rangle_{n,j}^{\mathsf{BCL}}, w_j)$ for a uniformly random $w_j \xleftarrow{\$} \mathbb{F}$. We recover K_j by the inverse function $K_j = w_j - \Delta \cdot \langle a \rangle_{n,j}^{\mathsf{BCL}}$ when emulating P_i's view (note that the ideal function output $\langle a \rangle_{n,j}^{\mathsf{BCL}}$ of corrupted party P_j is known to the simulator). Consequently, we do not make use of the private key Δ to simulate the views of corrupted parties.

As we discussed in the proof of Lemma 7.7, this change does not alter the view distribution and $\mathsf{Hyb}_2 \equiv \mathsf{Hyb}_3$.

Hybrid Ideal. This hybrid is the ideal world distribution. The simulator $\mathcal{S}$ works as follows:

- Sample Δ_i for every honest party P_i (such global secret key is never used to simulate the views of corrupted parties).
- Invoke a simulator $\mathcal{S}_{\mathsf{Ext}}$ of the protocol π_{Ext} to emulate the view of corrupted virtual parties, and emulate the behavior of the environment interacting with $\mathcal{S}_{\mathsf{Ext}}$. It corrupts the same set as the environment of π_{LTMPC} does.
- Emulate the protocol π_{LTMPC} as in Hyb_3, with the only exception of handling message transmission relevant to corrupted parties:
  - For any incoming message m from an honest V_i to a corrupted P_j, call S_{Ext} to provide the identity of m. In this way, $\mathcal{S}$ can generate a random BDOZ BCLSS $[m]_n^{\mathsf{BCL},i}$ as the output of $\mathcal{F}_{\mathsf{Int}}^i$.

– For any outgoing message m sent by a corrupted party V_i, $\mathcal{S}$ needs to determine the value of m. If the receiver V_j is also corrupted, it simply sends an all-0 message to $\mathcal{S}_{\mathsf{Ext}}$; otherwise, $\mathcal{S}$ captures the BDOZ BCLSS $[m]_n^{\mathsf{BCL},i}$ as input to $\mathcal{F}_{\mathsf{Com}}^{i,j}$, then sends m to $\mathcal{S}_{\mathsf{Ext}}$.

We have $\mathsf{Hyb}_3 \approx_S \mathsf{Ideal}$ because of the malicious security of π_{Ext}.
Consequently, we prove the following theorem:

Theorem 7.8. *Let $\ell = \ell(\kappa)$ be a leakage bound. Then for any $n \geq 2$, every n-party functionality $\mathcal{F}$ admits an unconditionally secure malicious $(n, \lfloor (n-1)/2 \rfloor, \ell)$-LTMPC with abort implementation, assuming honest parties can erase their local views.*

Remark 7.9 (On Semi-Honest Security)**.** If we settle for the weaker security notion of semi-honest security, we can simply discard the use of BDOZ MACs and substitute the protocol π_{Ext} with a simpler semi-honestly secure protocol (e.g. the BGW protocol [BOGW88]). We can argue the security of this approach by appropriately degenerating the current proof.

AI Disclosure

We thank ChatGPT 6 Astra for providing the impressive construction of the $(2, 0)$-LTOT protocol in the erasure model (Figure 20). The protocol was subsequently refined and streamlined by the authors into its current form, which uses LROTP as underlying primitive. ChatGPT is also widely applied to polish the writing and provide related literature. We have verified the correctness and originality of all content, including references.

References

[AJLA+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. In David Pointcheval and Thomas Johansson, editors, Advances in Cryptology – EUROCRYPT 2012, pages 483–501, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[AK25] Benny Applebaum and Eliran Kachlon. Nizk amplification via leakage-resilient secure computation. In Yael Tauman Kalai and Seny F. Kamara, editors, Advances in Cryptology – CRYPTO 2025, pages 462–479, Cham, 2025. Springer Nature Switzerland.

[BCG+11] Nir Bitansky, Ran Canetti, Shafi Goldwasser, Shai Halevi, Yael Tauman Kalai, and Guy N. Rothblum. Program obfuscation with leaky hardware. In Dong Hoon Lee and Xiaoyun Wang, editors, Advances in Cryptology – ASIACRYPT 2011, pages 722–739, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[BCH12] Nir Bitansky, Ran Canetti, and Shai Halevi. Leakage-tolerant interactive protocols. In Ronald Cramer, editor, Theory of Cryptography, pages 266–284, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, Advances in Cryptology – EUROCRYPT 2011, pages 169–188, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[BDSL14] Nir Bitansky, Dana Dachman-Soled, and Huijia Lin. Leakage-tolerant computation with input-independent preprocessing. In Juan A. Garay and Rosario Gennaro, editors, Advances in Cryptology – CRYPTO 2014, pages 146–163, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[Bea91] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology, CRYPTO '91, page 420–432, Berlin, Heidelberg, 1991. Springer-Verlag.

[BGJ+13] Elette Boyle, Sanjam Garg, Abhishek Jain, Yael Tauman Kalai, and Amit Sahai. Secure computation against adaptive auxiliary information. In Ran Canetti and Juan A. Garay, editors, Advances in Cryptology – CRYPTO 2013, pages 316–334, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[BGJK12] Elette Boyle, Shafi Goldwasser, Abhishek Jain, and Yael Tauman Kalai. Multiparty computation secure against continual memory leakage. In Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing, STOC '12, page 1235–1254, New York, NY, USA, 2012. Association for Computing Machinery.

[BH92] Donald Beaver and Stuart Haber. Cryptographic protocols provably secure against dynamic adversaries. In Rainer A. Rueppel, editor, Advances in Cryptology — EUROCRYPT' 92, pages 307–323, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, STOC '88, page 1–10, New York, NY, USA, 1988. Association for Computing Machinery.

[Bra87] Gabriel Bracha. An o(log n) expected rounds randomized byzantine generals protocol. J. ACM, 34(4):910–920, October 1987.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. J. Cryptol., 13(1):143–202, January 2000.

[Can01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, FOCS '01, page 136, USA, 2001. IEEE Computer Society.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgard. Multiparty unconditionally secure protocols. In Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, STOC '88, page 11–19, New York, NY, USA, 1988. Association for Computing Machinery.

[CDSMW09] Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. Improved non-committing encryption with applications to adaptively secure protocols. In Mitsuru Matsui, editor, Advances in Cryptology – ASIACRYPT 2009, pages 287–302, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology, CRYPTO '01, page 19–40, Berlin, Heidelberg, 2001. Springer-Verlag.

[CKL03] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In Proceedings of the 22nd International Conference on Theory and Applications of Cryptographic Techniques, EUROCRYPT'03, page 68–86, Berlin, Heidelberg, 2003. Springer-Verlag.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing, STOC '02, page 494–503, New York, NY, USA, 2002. Association for Computing Machinery.

[CPV17a] Ran Canetti, Oxana Poburinnaya, and Muthuramakrishnan Venkitasubramaniam. Equivocating yao: constant-round adaptively secure multiparty computation in the plain model. In Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, page 497–509, New York, NY, USA, 2017. Association for Computing Machinery.

[CPV17b] Ran Canetti, Oxana Poburinnaya, and Muthuramakrishnan Venkitasubramaniam. Better two-round adaptive multi-party computation. In Serge Fehr, editor, Public-Key Cryptography – PKC 2017, pages 396–427, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.

[DEOR04] Yevgeniy Dodis, Ariel Elbaz, Roberto Oliveira, and Ran Raz. Improved randomness extraction from two independent sources. In Klaus Jansen, Sanjeev Khanna, José D. P. Rolim, and Dana Ron, editors, Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, pages 334–344, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. SIAM J. Comput., 38(1):97–139, March 2008.

[FS89] Uriel Feige and Adi Shamir. Zero knowledge proofs of knowledge in two rounds. In Proceedings on Advances in Cryptology, CRYPTO '89, page 526–544, Berlin, Heidelberg, 1989. Springer-Verlag.

[GIM+16] Vipul Goyal, Yuval Ishai, Hemanta K. Maji, Amit Sahai, and Alexander A. Sherstov. Bounded-communication leakage resilience via parity-resilient circuits. In 2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS), pages 1–10, 2016.

[GIW17] Daniel Genkin, Yuval Ishai, and Mor Weiss. How to construct a leakage-resilient (stateless) trusted party. In Theory of Cryptography: 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part II, page 209–244, Berlin, Heidelberg, 2017. Springer-Verlag.

[GJS11] Sanjam Garg, Abhishek Jain, and Amit Sahai. Leakage-resilient zero knowledge. In Phillip Rogaway, editor, Advances in Cryptology – CRYPTO 2011, pages 297–315, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[GL02] Shafi Goldwasser and Yehuda Lindell. Secure computation without agreement. In Proceedings of the 16th International Conference on Distributed Computing, DISC '02, page 17–32, Berlin, Heidelberg, 2002. Springer-Verlag.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, STOC '87, page 218–229, New York, NY, USA, 1987. Association for Computing Machinery.

[Gol09] Oded Goldreich. Foundations of Cryptography: Volume 2, Basic Applications. Cambridge University Press, USA, 1st edition, 2009.

[GR12] Shafi Goldwasser and Guy N. Rothblum. How to compute in the presence of leakage. In Proceedings of the 2012 IEEE 53rd Annual Symposium on Foundations of Computer Science, FOCS '12, page 31–40, USA, 2012. IEEE Computer Society.

[IPS09] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In Omer Reingold, editor, Theory of Cryptography, pages 294–314, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[IS24] Yuval Ishai and Yifan Song. Leakage-tolerant circuits. In Marc Joye and Gregor Leander, editors, Advances in Cryptology – EUROCRYPT 2024, pages 196–225, Cham, 2024. Springer Nature Switzerland.

[IS25] Yuval Ishai and Yifan Song. Protecting computations against continuous bounded-communication leakage. In Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC '25, page 1887–1897, New York, NY, USA, 2025. Association for Computing Machinery.

[Lin09] Andrew Y. Lindell. Adaptively secure two-party computation with erasures. In Marc Fischlin, editor, Topics in Cryptology – CT-RSA 2009, pages 117–132, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[MS26] Yaohua Ma and Yifan Song. Leakage-tolerant circuits against ac0 leakage. In Nadia Heninger and Mike Rosulek, editors, Advances in Cryptology – CRYPTO 2026, pages 262–293, Cham, 2026. Springer Nature Switzerland.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. J. Cryptol., 4(2):151–158, January 1991.

[RBO89] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing, STOC '89, page 73–85, New York, NY, USA, 1989. Association for Computing Machinery.

[Sha79] Adi Shamir. How to share a secret. Commun. ACM, 22(11):612–613, nov 1979.

[Sho09] Victor Shoup. A Computational Introduction to Number Theory and Algebra. Cambridge University Press, USA, 2 edition, 2009.

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

A On the Impossibility of Secure Coin-Tossing

In this section, we rule out the possibility of realizing secure n-party coin-tossing protocols when exactly $n-1$ parties are maliciously corrupted. We note that our impossibility result does not coincide with the well-known impossibility result of UC-secure computation [Can01, CF01, CKL03] (in particular, UC-secure coin-tossing) *in the plain model*, because our proof can be directly extended to trusted setup models such as the common reference string (CRS) model.

Lemma A.1. *For any $n \geq 2$, there exists no maliciously secure $(n, n-1, 1)$-LTMPC protocol securely realizing the $\mathcal{F}_{\mathsf{coin-toss}}$ functionality that outputs a uniformly random bit b to all parties in either (1) the plain model, or (2) the CRS model.*

Proof. We only present the result in the plain model. Assume that there exists such a protocol $\pi_{\mathsf{coin-toss}}$ which proceeds in $T = \mathsf{poly}(\kappa)$ rounds to generate the randomness. Specifically, in the i-th round a predetermined party P_{c_i} generates a fresh random string $\boldsymbol{r}_i \xleftarrow{\$} \mathbb{F}_2^\kappa$. After the randomness generation phase, every honest party will agree on the bit $b \leftarrow F(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_T)$, where F is a deterministic function.

We then describe an adversary $\mathcal{A}$ that biases the outcome b toward 0 with non-zero advantage:

- $\mathcal{A}$ randomly picks an index $i \in [T]$, then corrupts every party except an arbitrary party $P^* \neq P_{c_i}$.
- $\mathcal{A}$ behaves honestly in the first $i-1$ rounds and records the randomness generated by the corrupted parties.
- In the i-th round, $\mathcal{A}$ first samples the randomness $\boldsymbol{r}_i \xleftarrow{\$} \mathbb{F}_2^\kappa$, then it examines whether this choice results in outcome $b = 0$ with high probability. Concretely, it randomly chooses $\boldsymbol{r}_{i+1}, \ldots, \boldsymbol{r}_T$, then launches a 1-bit leakage attack L on P^* which hardcodes $\boldsymbol{r}_{i+1}, \ldots, \boldsymbol{r}_T$ together with the randomness sampled by the corrupted parties in the first $(i-1)$ rounds. Since L can interact with the random choice made by P^* up to now, it can report the exact value of $b' \leftarrow F(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_T)$. If $b' = 0$, $\mathcal{A}$ continues to use $\boldsymbol{r}_i$ as the randomness in the i-th round; otherwise it simply picks another random string $\boldsymbol{r}'_i \xleftarrow{\$} \mathbb{F}_2^\kappa$.
- $\mathcal{A}$ also behaves honestly in the last $T-i$ rounds.

We claim that the advantage of $\mathcal{A}$ is at least $\frac{1}{T}(\frac{1}{4} - \mathsf{negl}(\kappa))$, which contradicts the security of $\pi_{\mathsf{coin-toss}}$. To see this, for a prefix $\mathsf{pre}_k = (\boldsymbol{r}_1, \ldots, \boldsymbol{r}_k)$ of the randomness, we denote $\Pr[F(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_T) = 0 : \boldsymbol{r}_{k+1}, \ldots, \boldsymbol{r}_T \xleftarrow{\$} \mathbb{F}_2^\kappa]$ as $p(\mathsf{pre}_k)$. We have $p(\emptyset) = \frac{1}{2} \pm \mathsf{negl}(\kappa)$ by the assumption, and $p(\mathsf{pre}_T) \in \{0, 1\}$ for every assignment of pre_T. Let $i^* \in [T]$ be any chosen index of $\mathcal{A}$. We analyze the advantage adv of $\mathcal{A}$ conditioned on a prefix pre_{i^*-1}:

$$\begin{aligned}
\mathsf{adv} &= \mathbb{E}_{r_{i^*} \xleftarrow{\$} \{0,1\}^\kappa}[p^2(\mathsf{pre}_{i^*-1} \parallel \boldsymbol{r}_{i^*}) + (1 - p(\mathsf{pre}_{i^*-1} \parallel \boldsymbol{r}_{i^*})) \cdot p(\mathsf{pre}_{i^*-1})] - p(\mathsf{pre}_{i^*-1}) \\
&= \mathbb{E}_{r_{i^*} \xleftarrow{\$} \{0,1\}^\kappa}[p^2(\mathsf{pre}_{i^*-1} \parallel \boldsymbol{r}_{i^*})] + (1 - p(\mathsf{pre}_{i^*-1})) \cdot p(\mathsf{pre}_{i^*-1}) - p(\mathsf{pre}_{i^*-1}) \\
&= \mathbb{E}_{r_{i^*} \xleftarrow{\$} \{0,1\}^\kappa}[p^2(\mathsf{pre}_{i^*-1} \parallel \boldsymbol{r}_{i^*})] - p^2(\mathsf{pre}_{i^*-1}).
\end{aligned}$$

Applying a telescoping step, we immediately obtain that the total advantage of $\mathcal{A}$ is:

$$\begin{aligned}
&\frac{1}{T}\left(\mathbb{E}_{\mathsf{pre}_T \xleftarrow{\$} (\mathbb{F}_2^\kappa)^T}[p^2(\mathsf{pre}_T)] - p^2(\emptyset)\right)\\
=&\frac{1}{T}\left(\mathbb{E}_{\mathsf{pre}_T \xleftarrow{\$} (\mathbb{F}_2^\kappa)^T}[p(\mathsf{pre}_T)] - p^2(\emptyset)\right)\\
=&\frac{1}{T}\left(p(\emptyset) - p^2(\emptyset)\right)\\
\geq&\frac{1}{T}\left(\frac{1}{4} - \mathsf{negl}(\kappa)\right),
\end{aligned}$$

□

where the first equality follows from the aforementioned fact $p(\mathsf{pre}_T) \in \{0,1\}$.

B 2-Source Extractor for BDOZ Triples

In this section, we introduce a 2-source extractor for BDOZ triples. To this end, our technical approach is to first design an extractor for Beaver triples and then enhance it into an extractor for BDOZ triples.

B.1 Beaver Triples Extractor

Let L be a parameter and set $(N, M, D, T) = (L^2+2L, 2L, 2L, L/2-2\log(1/\epsilon)-O(1))$. We present a $(N-T, M-T, D-T, \epsilon)$-2-source extractor for Beaver triples as follows:

- Parse $X = (U, V, O) \in \mathbb{F}_2^L \times \mathbb{F}_2^L \times \mathbb{F}_2^{L\times L}$ and $Z = (Z_L, Z_R) \in \mathbb{F}_2^L \times \mathbb{F}_2^L$. The functionality $\mathsf{Ext}(X, Z)$ outputs $(Z_L^T U, Z_R^T V, Z_L^T O Z_R)$.
- Parse $Y = (U, V) \in \mathbb{F}_2^L \times \mathbb{F}_2^L$. The Rand function simply outputs $(U, V, U \otimes V)$.

The linearity is easily verifiable, thus we focus on the randomness extraction and Beaver triples extraction properties.

Randomness Extraction. We prove a slightly stronger result that $\mathsf{Ext}(X, Z)$ is a 2-source extractor which is strong in the first input: viewing X as the seed, we claim that $\mathsf{Ext}(X, Z)$ is an approximate 2-universal hashing with a statistical distance $2^{-L/2}$ from uniform. Then the result follows from a standard collision probability bound on statistical distance (e.g., Shoup [Sho09, Theorem 8.36]), jointly applied to $(X, \mathsf{Ext}(X, Z))$:

$$\begin{aligned}
\Delta((X, \mathsf{Ext}(X, Z)), (X, U_3)) &\leq \frac{1}{2}\sqrt{2^{-\mathsf{H}_\infty(X)} \cdot 2^{N+3} \cdot 2^{-L/2}}\\
&\leq \frac{1}{2}\sqrt{2^{T-N} \cdot 2^{N+3} \cdot 2^{-L/2}}\\
&\leq \epsilon.
\end{aligned}$$

Lemma B.1. *For any random variable Z with min-entropy $\mathsf{H}_\infty(Z) \geq D - T$, $\mathsf{Ext}(X, Z)$ is an approximate 2-universal hashing if we take X as the seed. In other words,*

$$\Pr\left[\mathsf{Ext}(X, Z^1) = \mathsf{Ext}(X, Z^2) : Z^1, Z^2 \leftarrow Z, X \xleftarrow{\$} U_N\right] \leq \frac{1}{8} + 2^{-L/2}.$$

Proof. For any pair $(Z^1, Z^2) \in (\mathbb{F}_2^D)^2$, we observe that if $Z_L^1 \neq Z_L^2$, $Z_R^1 \neq Z_R^2$ and $Z_L^1 \otimes Z_R^1 \neq Z_L^2 \otimes Z_R^2$,

$$\Pr[\mathsf{Ext}(X, Z^1) = \mathsf{Ext}(X, Z^2) : X \xleftarrow{\$} U_N] = \frac{1}{8}.$$

Furthermore, if none of $Z_L^1, Z_R^1, Z_L^2, Z_R^2$ equals 0^L, then $Z_L^1 \otimes Z_R^1 \neq Z_L^2 \otimes Z_R^2$ holds as long as $Z_L^1 \neq Z_L^2$ and $Z_R^1 \neq Z_R^2$. By a union bound, we only need to bound the probabilities of the events $Z_L^1 = Z_L^2$ and $Z_R^1 = Z_R^2$ separately. By a direct calculation, we have

$$\begin{aligned}
\Pr[Z_L^1 = Z_L^2] &\leq 2^{-\mathsf{H}_\infty(Z_L)} \\
&\leq 2^{-\mathsf{H}_\infty(Z_L, Z_R)+|Z_R|} \\
&\leq 2^{T-L} \\
&\leq 2^{-L/2-3}.
\end{aligned}$$

Similarly, we also have $\Pr[Z_R^1 = Z_R^2] \leq 2^{-L/2-3}$, indicating that

$$\begin{aligned}
&\Pr[\mathsf{Ext}(X, Z^1) = \mathsf{Ext}(X, Z^2)] \\
\leq &\frac{1}{8} + \Pr[Z_L^1 = Z_L^2] + \Pr[Z_R^1 = Z_R^2] + \Pr[Z_L^1 = 0 \vee Z_L^2 = 0 \vee Z_R^1 = 0 \vee Z_R^2 = 0] \\
\leq &\frac{1}{8} + 2 \cdot 2^{-L/2-3} + 2 \cdot 2^{-\tilde{\mathsf{H}}_\infty(Z_L|Z_R)} + 2 \cdot 2^{-\tilde{\mathsf{H}}_\infty(Z_R|Z_L)} \\
\leq &\frac{1}{8} + 6 \cdot 2^{-L/2-3} \\
\leq &\frac{1}{8} + 2^{-L/2}.
\end{aligned}$$

□

Beaver Triples Extraction. The correctness is straightforward. To prove that $\mathsf{Ext}(\mathsf{Rand}, \cdot)$ is indeed a 2-source Beaver triples extractor, we only need to show that

$$\Delta((Z_L^T U, Z_R^T V), U_2) \leq \epsilon,$$

which is a simple application of [DEOR04, Theorem 1].

B.2 BDOZ Triples Extractor

Let $\Delta \in \mathbb{F}$ be the global secret key and denote $\mathbb{F}' = \mathbb{F}_2 \times \mathbb{F}$. To compile the aforementioned 2-source extractor for Beaver triples into an extractor for BDOZ triples, we introduce the following functions related to MAC extraction:

- Parse $X = (X_i \in \mathbb{F}_2)_{i \in [N]}$ and $KS = (KS_i \in \mathbb{F})_{i \in [N]}$. Then $\mathsf{Sign}_\Delta(X, KS)$ outputs the concatenation of bitwise MACs $\mathsf{MAC}_{(\Delta, KS_i)}(X_i)$. Specifically, we have

  $$\begin{aligned}
  \mathsf{Sign}_\Delta(X, KS) &= (\mathsf{MAC}_{(\Delta, KS_i)}(X_i))_{i \in [N]} \\
  &= (X, KS + \Delta \cdot X).
  \end{aligned}$$

  The linearity of Sign_Δ follows from the linearity of BDOZ MACs. Moreover, for a fixed pair of (Δ, X), $(X, W) \leftarrow \mathsf{Sign}_\Delta(X, KS)$ is a translation of KS, thus we can restore $KS = W - \Delta \cdot X$ from W.

- Parse $KS = (KSU, KSV, KSO) \in \mathbb{F}^L \times \mathbb{F}^L \times \mathbb{F}^{L\times L}$ and $Z = (Z_L, Z_R) \in \mathbb{F}_2^L \times \mathbb{F}_2^L$. The functionality $\mathsf{KeyExt}_\Delta(KS, Z)$ simply outputs $((\Delta, Z_L^T \cdot KSU), (\Delta, Z_R^T \cdot KSV), (\Delta, Z_L^T \cdot KSO \cdot Z_R))$.

  The inverse function of KeyExt_Δ can be explicitly constructed as follows: given a random Z s.t. $Z_L, Z_R \neq 0^L$ (which happens with overwhelming probability), $(K_a, K_b, K_c) \leftarrow \mathsf{KeyExt}_\Delta(KS, Z)$ uniformly maps $KS \in \mathbb{F}^N$ to $(K_a, K_b, K_c) \in (\mathbb{F}^2)^3$ (conditioned on the global secret key Δ). W.L.O.G. assume that $(Z_L)_1 \neq 0$ and $(Z_R)_1 \neq 0$. Then, for any assignment of

  $$((KSU)_{[L]/\{1\}}, (KSV)_{[L]/\{1\}}, (KSO)_{([L]\times[L])/\{(1,1)\}}),$$

  (K_a, K_b, K_c) is an affine function of $(KSU_1, KSV_1, KSO_{(1,1)})$ and can be efficiently inverted.

- Parse $(X, W) = ((U, WU), (V, WV), (O, WO)) \in \mathbb{F}'^L \times \mathbb{F}'^L \times \mathbb{F}'^{L\times L}$. The functionality MACExt outputs

  $$(Z_L^T \cdot (U, WU), Z_R^T \cdot (V, WV), Z_L^T \cdot (O, WO) \cdot Z_R).$$

  We can verify that for $(X, W) \leftarrow \mathsf{Sign}_\Delta(X, KS)$, $(a, b, c) \leftarrow \mathsf{Ext}(X, Z)$ and $(K_a, K_b, K_c) \leftarrow \mathsf{KeyExt}_\Delta(KS, Z)$:

  $$\begin{aligned}
  \mathsf{MACExt}((X, W), Z) &= (Z_L^T \cdot (U, WU), Z_R^T \cdot (V, WV), Z_L^T \cdot (O, WO) \cdot Z_R) \\
  &= (Z_L^T \cdot (U, KSU + \Delta \cdot U), Z_R^T \cdot (V, KSV + \Delta \cdot V), Z_L^T \cdot (O, KSO + \Delta \cdot O) \cdot Z_R) \\
  &= ((a, Z_L^T \cdot KSU + \Delta \cdot a), (b, Z_R^T \cdot KSV + \Delta \cdot b), (c, Z_L^T \cdot KSO \cdot ZR + \Delta \cdot c)) \\
  &= (M(a), M(b), M(c)).
  \end{aligned}$$

  Clearly, MACExt also admits a simple inverse function like that of KeyExt_Δ when $Z_L, Z_R \neq 0^L$.