NIZK Amplification via Leakage-Resilient Secure Computation*

Benny Applebaum† Eliran Kachlon*

May 29, 2025

Abstract

Suppose that we are given a weak *Non-Interactive Zero-Knowledge* (NIZK) proof system for **NP** with non-negligible soundness and zero-knowledge errors, denoted by α and β, respectively. Is it possible to to reduce these errors to a negligible level? This problem, known as NIZK amplification, was introduced by Goyal, Jain, and Sahai (Crypto'19) and was further studied by Bitansky and Geier (Crypto'24). The latter work provides amplification theorems for proofs and arguments, assuming the existence of one-way functions and public-key encryption, respectively. Unfortunately, their results only apply when the security level, $1-(\alpha+\beta)$, is a constant bounded away from zero. Amplifying NIZK with an inverse polynomial security level remains an open problem and was stated as the main open question in both works.

In this work, we resolve the NIZK amplification problem and show how to amplify any non-trivial NIZK proof system that has a noticeable, inverse-polynomial level of security. As in previous works, we amplify proofs and arguments assuming the existence of one-way functions and public-key encryption, respectively. Furthermore, assuming the existence of collision-resistant hash functions, we preserve, for the first time, properties such as statistical zero-knowledge and proof succinctness.

Our main technical contribution is a new *leakage-resilient secure multiparty* protocol that computes any public-output functionality with information-theoretic security against an adversary that corrupts an arbitrary subset of parties and obtains bounded leakage from each honest party. Our protocol operates in the pairwise correlated randomness model. Previous works relied on stronger setup assumptions in the form of n-wise correlations and either supported a smaller corruption threshold or suffered from an exponential dependency on the number of parties. To transform our protocol into a NIZK amplifier, we introduce a new intermediate notion of *leakage-resilient NP secret sharing*, that may be of independent interest.

*This is the full version of a a paper that will appear in the proceedings of Crypto'25 conference.

†Tel-Aviv University, Israel bennyap@post.tau.ac.il, elirn.chalon@gmail.com. Supported by ISF grant no. 2805/21 and by the European Union (ERC-2022-ADG) under grant agreement no.101097959 NFITSC.

Contents

1 Introduction **4**
- 1.1 Main Result: NIZK Amplification . . . 5
- 1.2 Leakage-Resilient MPC . . . 6
- 1.3 Leakage-Resilient NPSS . . . 8

2 Technical Overview **9**
- 2.1 Leakage-Resilient MPC . . . 9
  - 2.1.1 Two-Party LRMPC . . . 9
  - 2.1.2 From Two Parties to Multiple Parties . . . 9
  - 2.1.3 Compiling OT Calls . . . 10
  - 2.1.4 Pairwise Correlations . . . 11
  - 2.1.5 Threshold LRMPC . . . 11
- 2.2 NIZK Amplification . . . 13
  - 2.2.1 The First Amplifier . . . 13
  - 2.2.2 The Second Amplifier: Reducing the ZK Error to Negligible . . . 15

3 MPC: Basic Definitions **16**
- 3.1 Formalizing Protocols . . . 16
- 3.2 Security Definition . . . 17

4 Leakage Resilient MPC: Basic Definitions **19**
- 4.1 Security Definition . . . 19
- 4.2 The Secret Sharing Scheme . . . 20
- 4.3 Leakage-Resilient Composable Protocols . . . 22
  - 4.3.1 On Two-Party Composable Protocols . . . 23
  - 4.3.2 On Senders-Receivers Composable Protocols . . . 24
  - 4.3.3 Strongly-Composable Two-Party Protocols . . . 25

5 Leakage-Resilient MPC: Basic Construction **27**
- 5.1 Compiling Protocol Instructions . . . 27
  - 5.1.1 Compiling Local Operations . . . 28
  - 5.1.2 Compiling Output Statement . . . 28
  - 5.1.3 Compiling Abort Statement . . . 29
  - 5.1.4 Compiling OLE Statements . . . 29
- 5.2 The Compiler . . . 32

6 Leakage Resilient MPC: Pairwise Correlations **37**
- 6.1 Private Simultaneous Message Protocols . . . 38
- 6.2 Our Construction . . . 38
- 6.3 Instantiation: n-out-of-n Leakage-Resilient MPC with Pairwise Correlations . . . 41

7 NPSS and Leakage-Resilient NPSS **42**
- 7.1 NPSS: Definition . . . 42
- 7.2 From Leakage-Resilient MPC to Leakage-Resilient NPSS . . . 43
- 7.3 Constructing t-out-of-n Leakage-Resilient NPSS . . . 46

8 Application: NIZK Amplification **47**
8.1 Useful Primitives and Lemmas . . . 48
8.1.1 Non-Interactive Commitments and Trapdoor Commitments . . . 48
8.1.2 XOR Lemmas and Guessing Games . . . 50
8.1.3 Indistinguishability Lemma . . . 51
8.1.4 Basic NIZK Transformations . . . 53
8.2 Main Amplifier: Negligible Soundness Error and Inverse-Polynomial WI Error . . . 53
8.2.1 Amplification Lemmas . . . 54
8.2.2 Proof of Lemma 8.14 . . . 55
8.2.3 Proof of Lemma 8.15 . . . 61
8.2.4 Proof of Lemma 8.16 . . . 63
8.3 From Inverse-Polynomial to Negligible WI Error . . . 63
8.3.1 Proof of Lemma 8.21 . . . 65
8.4 Putting it All Together . . . 69

A Preliminaries **76**
A.1 Statistical Distance and Entropy . . . 76
A.2 Cryptographic Primitives . . . 76
A.2.1 Non-Interactive Zero-Knowledge . . . 77
A.2.2 Public-Key Encryption . . . 78

B Missing Proofs: Leakage Resilient MPC **79**
B.1 Proof of Theorem 4.11 . . . 79
B.2 Proof of Lemma 5.2 . . . 80
B.2.1 The Simulator . . . 81
B.2.2 Analysis . . . 82
B.2.3 Proof of Claim B.2 . . . 83
B.2.4 Proof of Claim B.3 . . . 84
B.2.5 Proof of Claim B.5 . . . 88
B.2.6 Proof of Claim B.6 . . . 88
B.3 Proof of Lemma 5.7 . . . 89
B.3.1 Statistical Privacy . . . 89
B.4 Proof of Lemma 5.8 . . . 94
B.5 Proof of Lemma 6.4 . . . 95

C Missing Proofs: NIZK **98**
C.1 Proof of Theorem 8.5 . . . 98
C.2 Proof of Lemma 8.11 . . . 100
C.3 Proof of Lemma 8.12 . . . 101
C.4 Proof of Lemma 8.13 . . . 102

1 Introduction

The problem of *security amplification* is a cornerstone of modern cryptography, and has been extensively studied in numerous contexts (e.g., [Yao82, Imp95, GNW11, DKS99, GP99, DNR04, Hol05, Wul07, HR08, IJK09, MT10, LT13]). Amplification procedures are crucial for the study of the relations between different primitives (i.e., "structural cryptography"), and play a key role in cryptographic constructions allowing the designer to focus on a "weak" primitive and upgrade it into a "strong" one.

Quantitatively, weak typically means that the *security error* of the cryptographic primitive (i.e., the best success probability or advantage of an attacker with polynomial resources) is bounded away from 1 by some inverse polynomial quantity (the adversary fails with noticeable probability), and strong means that the security error is negligible. Two canonical examples are Yao's amplification of one-way functions [Gol01, Chapter 2.3] and Yao's XOR lemma [GNW11] that can be used to amplify the security of pseudorandom generators. As demonstrated by these results, amplifying search games is typically much simpler than amplifying distinguishing games.

In this paper we consider the problem of NIZK Amplification [GJS19, BG24]. An (α, β)-weak NIZK is a NIZK for an **NP**-complete language with a soundness error of at most α and a zero-knowledge error, measured by distinguishing advantage, is at most β.[1] When the total *security error* $\alpha + \beta$ is negligible, we recover the standard notion of NIZK. At the other extreme, when the security error $\alpha + \beta$ equals to 1, the definition becomes trivial: a prover could satisfy the definition by simply outputting the witness with probability β and an empty string with probability α, where the choice can be determined by the CRS or by the private coins of the prover. Therefore, the interesting non-trivial, case lies in the intermediate regime where the security error $\alpha + \beta$ is bounded-away from 1 by an inverse-polynomial quantity $1/p(\kappa)$. (Here and throughout the paper κ plays the role of a security parameter.) Specifically, [GJS19] asked whether any non-trivial NIZK for **NP** can be amplified into a fully secure NIZK. The setting of NIZK amplification is challenging since both soundness and zero knowledge need to be simultaneously amplified. Furthermore, the failure of zero-knowledge may be correlated with the witness that is being used.

Amplifier vs. combiner. To better understand the last point, let us first consider the easier task of constructing a NIZK *combiner*. A NIZK combiner takes as an input n NIZK candidates with the guarantee that soundness holds for all but a of the candidates, while zero knowledge holds for all but b of the candidates, and the goal of the combiner is to output a single NIZK construction that satisfies both soundness and zero knowledge with negligible error. It is not hard to see that a NIZK amplifier implies a NIZK combiner. Indeed, by picking a NIZK candidate at random we obtain an $(a/n, b/n)$-weak NIZK that we can then amplify.

Naively, one could think that the inverse is also true, i.e., that a NIZK combiner implies a NIZK amplifier, since given an (α, β)-weak NIZK Π we could take a sufficiently large number n of independent copies of Π, and argue that, with overwhelming probability, soundness holds in $(\alpha + \frac{1}{p(\kappa)})$-fraction of the copies, while zero knowledge holds in $(\beta + \frac{1}{p(\kappa)})$-fraction of the copies, for some arbitrarily small polynomial p. However, this intuition turns out to be false, since it is possible that zero knowledge does not hold in *any* of the candidates. As a concrete example, think of a weak NIZK that always reveals the first bit of the secret witness (and nothing else). Such a

[1] Throughout the paper, we fix circuit-SAT as a canonical **NP**-complete language, and refer to "NIZK for circuit-SAT" simply as NIZK.

weak NIZK has a zero-knowledge error of $1/2$ since we can always let the simulator "guess" the first bit of the witness. In particular, none of the n copies satisfies zero knowledge, and therefore the result will not necessarily satisfy zero knowledge.

Previous works. The question of NIZK amplification was first raised by [GJS19], that also suggested an elegant approach for the case where the security error $(\alpha + \beta)$ is bounded-away from 1 by some positive constant. Their idea was to sample many instances NIZK (with independent common reference strings) and iterate between a soundness amplification step (based on parallel repetition) and a zero-knowledge amplification step. To achieve the latter task they show how to map a circuit-SAT instance f and a satisfying assignment x into n instance/witness pairs (f_i, y_i) such that their joint satisfiability implies the satisfiability of f but any strict-subset of the witnesses reveal no information on the original assignment x. This notion of n-out-of-n *secret sharing for* **NP**-*statement* (NPSS) was realized by [GJS19] with computational security and was used in the natural way to amplify zero-knowledge. Unfortunately, their amplifier was found to be flawed (as noted by the authors [Jai24]). In particular, n-out-of-n NPSS does not deal with the case where each NIZK-instance leaks some information on the witness.

In a followup work, Bitanksy and Geier [BG24] recovered the statement claimed by [GJS19] and provided a NIZK amplification theorem, again for the case where the security error is bounded-away from 1 by a constant. Their solution is based on an indirect approach: First use the weak-NIZK to realize the so-called "hidden bit model" (HBM) and then use this model to realize NIZK via the celebrated FLS construction [FLS90]. The main insight is that realizing HBM can be reduced to proving **NP**-statements about pseudorandom generators, and by using clever pseudorandomness amplification ideas (e.g., randomness extractors) one can make such statements resilient to the leakage incurred by the weak zero-knowledge property of NIZK. Their amplification theorem relies on weak assumptions. Namely, one-way functions if the underlying NIZK has statistical soundness, and public-key encryption when it has computational soundness. On the downside, their theorem inherently downgrades zero-knowledge to computational (even if the underlying NIZK has statistical zero-knowledge properties) and does not preserve succinctness (due to the use of FLS).

In a companion paper [AK25] by the authors of the current paper, the notion of NPSS is further explored. This includes new refined definition of NPSS, information-theoretic constructions of t-out-of-n NPSS for arbitrary values of t and n, and several applications of NPSS as a non-interactive combiner for zero-knowledge proofs in various settings. While [AK25] shows that NPSS is a powerful tool, the paper makes no progress in the question of amplifying NIZKs.

To summarize, the question of NIZK-amplification in the inverse polynomial regime remains an intriguing open question, as noted in both [BG24] and the CRYPTO'19 talk by [GJS19]. Furthermore, it is currently unknown whether it is possible to amplify NIZK while preserving statistical zero-knowledge and/or succinctness, even for simple cases where the sum of the initial errors, $\alpha + \beta$ is bounded away from 1 by a constant, e.g., $\alpha = \beta = 0.4$.

1.1 Main Result: NIZK Amplification

We resolve the problem of NIZK amplification and prove the following result.

Theorem 1.1 (Informal). *Let Π_{wk} be an (α, β)-weak NIZK, where*

$$\alpha(\kappa) + \beta(\kappa) < 1 - \frac{1}{\mathrm{poly}(\kappa)}.$$

Then,

1. *If Π_{wk} satisfies α-statistical-soundness and β-computational-zero-knowledge, then assuming the existence of one-way functions there exists a NIZK Π with statistical soundness and computational zero knowledge.*

2. *If Π_{wk} satisfies α-computational-soundness and β-computational-zero-knowledge, then assuming the existence of public-key encryption there exists a NIZK Π with computational soundness and zero knowledge.*

3. *If Π_{wk} satisfies α-proof-of-knowledge and β-statistical-zero-knowledge, then assuming the existence of collision-resistant hash functions there exists a NIZK Π with proof of knowledge and statistical zero knowledge. Furthermore, the transformation preserves succinctness in the sense that if the proof length in Π_{wk} for S-size circuits is $\mathrm{poly}(\kappa, \log(S))$ then so is Π.*

To prove Theorem 1.1, we introduce a new notion of *leakage-resilient NPSS* (LRNPSS) and show that it can be used for NIZK amplification. This transformation turns to be much more subtle and technical than the one used in the context of NIZK-combiners. We then reduce LRNPSS to the problem of leakage-resilient secure multiparty computation (LRMPC), and show how to realize such LRMPC. Along the way, we construct the first information-theoretic LRMPC that remains secure against arbitrary coalitions and uses only *pair-wise* correlations. We continue with a detailed acount of these results.

1.2 Leakage-Resilient MPC

Recall that a t-private MPC protocol allows to securely compute an n-party functionality f without leaking any information to coalitions of size at most t. For now, we focus on the setting of a passive (aka semi-honest) adversary and so require that the view of every t-subset of corrupted parties can be simulated. In the information-theoretic setting where the simulation should be statistically close the actual view, this can be achieved unconditionally either in the standard model [BGW88, CCD88] when $t < n/2$ or in the OT-hybrid model [GMW87] for arbitrary $t \leq n$. The latter result can be also stated in the following *pairwise correlated randomness* model. In an offline preprocessing step (before the inputs are known), a trusted party samples, for each pair of parties, (i, j), a pair of strings $(\mathbf{r}^i_{i,j}, \mathbf{r}^j_{i,j})$ that are sampled from some joint distribution $\mathsf{Rand}_{i,j}$ whose length may depend on the circuit size of f. Later in the online phase, the parties get their inputs and communicate via standard (possibly public) channels. In actual implementations, the offline step can be realized via pairwise secure multiparty protocols or via some other secure service.

In the leakage-resilient setting [BGK14, BGJK12, GR15, BDSL14, GIM+16], the adversary obtains the entire view of the corrupted parties, as well as a bounded leakage from each view of the honest parties. (Imagine that a virus corrupts the honest parties and sends a bounded amount of information to the adversary). Formally, the adversary should specify the set of corrupted parties Z (of size at most t) and a list of computationally-unbounded leakage functions $(\tau_i)_{i \notin Z}$ each outputting at most μ bits. After an honest execution in which the view of the ith party is v_i the

adversary receives $(v_i)_{i\in Z}$ and $(\tau_i(v_i))_{i\notin Z}$, and these values should be simulated up to some small *privacy error*.

Observe that the leakage functions may reveal some information about the inputs and outputs of the honest parties. To address this issue, we follow the standard approach (see, e.g., [BDIR21]) and assume that the input $\mathbf{x}$ is encoded before the execution. That is, we assume that there is a leakage-free input encoder Enc that transforms the input $\mathbf{x}$ into a distributed encoding $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$, where the i-th party takes $\mathbf{x}_i$ as an input. To ensure that the encoder does not trivialize the problem, we require its complexity to depend logarithmically on the size of the circuit f being computed. We focus on the case in which the functionality f delivers the same output to all the parties, and therefore we may assume that the output is given to the adversary. (This corresponds to "public output" model of [BDIR21], our protocols can be easily adopted to their private output model as well.)

Our first result is a leakage-resilient version of the GMW theorem. That is, we construct an MPC protocol in the *pairwise correlations* model with information-theoretic security against an adversary that corrupts up to $n-1$ of the parties and, for a given statistical security parameter κ, can leak $\mu = \log(\kappa)$ bits from each honest party. The protocol achieves an inverse polynomial *statistical* privacy error of $1/\kappa$. We leave it as an interesting open question to reduce the leakage error to negligible.

Theorem 1.2 (Informal)**.** *For every public-output functionality f, security parameter κ, and number of parties n, there is an n-party leakage-resilient MPC where the adversary can corrupt $n-1$ of the parties. The protocol has pairwise correlations, leakage parameter $\mu = \log(\kappa)$ and privacy error $1/\kappa$. The complexity of the protocol is $\mathrm{poly}(n, \kappa, |f|)$.*

Comparison with previous works. LRMPC protocols for the setting of many parties were given in [BDIR21, SV19]. Both works achieve information-theoretic security with better leakage parameters but rely on a stronger set-up at the form of *n-wise correlated randomness* and cannot handle arbitrary coalitions when the number of parties is large (not constant). In the model of n-wise correlated randomness the offline trusted party jointly samples n random strings $(\mathbf{r}_i)_{i\in[n]}$ whose length is linear in the circuit size of f, and delivers $\mathbf{r}_i$ to the ith party. This model is considered to be significantly stronger (and less realistic) than the pairwise correlated randomness model. In particular, in order to generate such randomness one has to invoke a global procedure that simultaneously involves all the parties. (See also the discussion in [FGMO05])

In more detail, the first LRMPC protocol with information-theoretic security for a large number of parties was constructed in [BDIR21]. For the case of n parties, their protocol tolerates $O(\log(n))$ bits from each party, and the privacy error is $2^{-\Omega(n)}$. On the downside, they allow the adversary to corrupt only an α-fraction of the parties for some constant $\alpha < 1$.[2] The work of [SV19] also presents leakage-resilient MPC protocols, although their main focus is on general interaction patterns. In the information-theoretic settings, they describe two constructions that tolerate κ bits of leakage from the honest parties while providing a privacy error of $2^{-\kappa}$. On the downside, their first construction suffers from an exponential complexity in the number of inputs and parties (compared to polynomial in our case), while their second construction is limited to $\mathbf{NC}^1$ functions and limit the adversary to corrupt only a constant number of parties. As already mentioned, all

[2] In their theorem statements $\alpha < 0.5$, however, it seems that the same techniques can achieve a larger constant (bounded away from 1).

these constructions rely on n-wise correlated randomness. (We refer the reader to [BDIR21] for a detailed overview of LRMPC protocols in the setting of constant number of parties.)

1.3 Leakage-Resilient NPSS

As already mentioned, a *t-out-of-n secret sharing of an* **NP** *statement* (NPSS) is a reduction that maps an instance/witness pair into n instance/witness pairs such that any set of $(t-1)$ of them reveals no information on the original witness and any t out of them allow to fully recover the original witness. Following the recent formulation of [AK25], our NPSS definition is composed of four efficient algorithms: a deterministic instance mapper R, a randomized assignment mapper W, a randomized simulator Sim and a deterministic decoder Dec. Given a circuit-SAT instance f, the instance mapper deterministically maps f into n circuit-SAT instances $F = (f_1, ..., f_n)$ over a *common set of variables* such that the followings hold:

- *Correctness and $(t-1)$-privacy:* Given a satisfying assignment x for f, the randomized assignment mapper W samples n partial assignments $(y_1, \ldots, y_n)$ that *consistently satisfy* $(f_1, ..., f_n)$ in the sense that y_i satisfies f_i and there exists some global assignment y that agrees with all the partial assignments. Furthermore, the marginal distribution of every collection Z of $t-1$ partial assignments $(y_{i_1}, \ldots, y_{i_{t-1}})$ leaks no information about the witness x, i.e., the joint distribution can be efficiently sampled based on f via the simulator $\mathsf{Sim}(f, \{i_1, \ldots, i_{t-1}\})$.
- *t-recovery:* Given any collection of t partial assignments $y_{i_1}, \ldots, y_{i_t}$ that consistently satisfy $f_{i_1}, \ldots, f_{i_t}$, the decoder Dec efficiently recovers an assignment x that satisfies f. This implies a strong *soundness* property: if f is unsatisfiable, it is impossible to consistently satisfy more than t of the instances in F.

Note that correctness, privacy and recovery hold information-theoretically. In the context of NIZK amplification, we will have to strengthen the privacy property to holds even if some information is leaked on the witnesses $(y_i)_{i \notin Z}$ that are not given to the adversary. As in the LRMPC setting, this is captured by allowing the adversary to specify a list of computationally-unbounded leakage functions $(\tau_i)_{i \notin Z}$ each outputting at most μ bits. Roughly speaking, we say that the NPSS is (μ, ϵ) Leakage Resilient (or (μ, ϵ)-NPSS in short) if the distribution $(y_i)_{i \in Z}, (\tau_i(y_i)_{i \notin Z})$ can be simulated within a statistical distance of at most ϵ.

Standard NPSS (without leakage resiliency) were constructed in [AK25] based on MPC protocols with some non-standard correctness-under-active corruptions property (to be discussed later). We extend the LRMPC protocols constructed in the previous section to satisfy this additional requirement, and transform them into LRNPSS. This yields the following theorem.

Theorem 1.3. *There is a deterministic compiler C_{Th} that takes as an input two integers $2 \leq t \leq n$, runs in time $\mathrm{poly}(n)$ and outputs the description of a t-out-of-n $(\mu(\kappa) = \lfloor \log(\kappa) \rfloor, \epsilon(\kappa) = 1/\kappa)$-NPSS $(R, W, \mathsf{Sim}, \mathsf{Dec})$. The running time of all algorithms in $(R, W, \mathsf{Sim}, \mathsf{Dec})$ is $\mathrm{poly}(\kappa \cdot n \cdot |f|)$ where f is the circuit-SAT instance that the algorithms take as an input, and $|f|$ is the circuit-size of f measured by the number of wires.*

2 Technical Overview

2.1 Leakage-Resilient MPC

In this section we present an overview of our leakage-resilient MPC protocol. We first describe a protocol with calls to two-party ideal functionalities and without correlated randomness, and then explain how to replace those ideal functionalities with sub-protocols that require only pairwise correlations. It will be instructive to first consider the basic case of *two-party* leakage-resilient MPC.

2.1.1 Two-Party LRMPC

When leakage resiliency is not required, one could simply execute the two-party protocol of GMW [GMW87]. Recall that in the GMW protocol, the inputs are first shared among the parties, and then the parties compute the circuit gate-by-gate in topological ordering, where for each gate, given a secret sharing of the inputs, they compute a secret sharing of the output with the aid of an oblivious transfer.

Our protocol follows the same high-level approach of the GMW protocol. That is, first we let the input encoder share the input $\mathbf{x}$ among Alice and Bob. The input is shared bit-by-bit using a variant of the leakage-resilient secret-sharing (LRSS) scheme of [SV19]. To compute the circuit, we note that the LRSS of [SV19] satisfies the following special property: for every bit $s \in \{0,1\}$, we can let Alice sample a random string $\mathbf{s}_A$ as a share of s, we can let Bob sample a (slightly shorter) random string $\mathbf{s}'_B$ as a "partial share", and there exists a deterministic procedure $\mathsf{LRUpdate}$ that given $(\mathbf{s}_A, \mathbf{s}'_B, s)$ outputs u_B, such that the shares $(\mathbf{s}_A, \mathbf{s}_B := (\mathbf{s}'_B, u_B))$ form a fresh leakage-resilient secret sharing of s.

To compute any gate $* \in \{+, \times\}$ that maps inputs x, y to an output z, the parties call an ideal functionality that takes the shares of the inputs, $(\mathbf{x}_A, \mathbf{x}_B)$ and $(\mathbf{y}_A, \mathbf{y}_B)$, as well as a share and partial share that corresponds to the output $(\mathbf{z}_A, \mathbf{z}'_B)$, recovers the inputs x and y, computes the outputs $z := x * y$ and outputs $u_B = \mathsf{LRUpdate}(\mathbf{z}_A, \mathbf{z}'_B, z)$ to Bob. As a result, the shares $(\mathbf{z}_A, \mathbf{z}_B = (\mathbf{z}'_B, u_B))$ are fresh shares of z, as required. Overall, the view of each party contains only a fresh (leakage-resilient) share for each circuit's wire, and, by a standard hybrid argument, the privacy error grows linearly with the circuit's size s. That is, if the privacy error of the LRSS is ϵ, then the total privacy error is $s \cdot \epsilon$.

2.1.2 From Two Parties to Multiple Parties

To obtain an n-party leakage-resilient protocol for the computation of f, we use the virtualization technique of [HM00, CDI+13]. Assume for simplicity that n is even. The high-level idea is to take a protocol π' with $n/2$ parties that satisfies standard (non-leakage-resilient) $(n/2-1)$-privacy, and compile it into a leakage-resilient n-party protocol π with security against any adversary that corrupts $n-1$ of the parties. In more details, we take π' to be the GMW protocol in the OT-hybrid model, that allows $n/2$ parties $w_1, \ldots, w_{n/2}$ that hold additive secret sharing of $\mathbf{x}$, denoted $(\mathbf{x}^1, \ldots, \mathbf{x}^{n/2})$, to recover the output $f(\mathbf{x})$. We compile π' into an n-party leakage-resilient protocol π by using player virtualization: we replace each party w_i with two parties, $p_{i,0}$ and $p_{i,1}$, that emulate w_i's behavior in an execution of π' while only holding an LRSS of w_i's view. The LRSS guarantees that even when the adversary corrupts all the parties except for some $p_{i,b}$, the total

knowledge of the adversary (i.e., the views of all corrupt parties and the leakage from $p_{i,b}$) reveals no information about the view of the virtual player w_i, and, in particular, on the share $\mathbf{x}^i$ of $\mathbf{x}$. Privacy now follows from the privacy of π'. Put differently, we can think of the virtual player w_i as "honest" if either $p_{i,0}$ or $p_{i,1}$ is honest.

We continue by explaining how to emulate the w's. First, we let the input encoder of π take the input $\mathbf{x}$, compute additive secret sharing $(\mathbf{x}_1, \ldots, \mathbf{x}_{n/2})$ of $\mathbf{x}$, and share $\mathbf{x}_i$ among $p_{i,0}$ and $p_{i,1}$ using LRSS, which means that $p_{i,0}$ and $p_{i,1}$ indeed hold an LRSS of w_i's input. Then, the parties $p_{1,0}, p_{1,1}, \ldots, p_{n/2,0}, p_{n/2,1}$ emulate the views of the w's in an execution of π': Whenever w_i is instructed by π' to perform some local computation comp, in π we let $p_{i,0}$ and $p_{i,1}$ execute our two-party LRMPC protocol to compute an LRSS of the result of comp. However, when w_i is instructed to send a private message to w_j, or when w_i and w_j are instructed to perform an OT call, things get more complicated since the compilation now involves *four* parties: the two emulating w_i and the two emulating w_j. Since sending private messages can be reduced to OT calls, it suffices to explain how to compile OT calls.

2.1.3 Compiling OT Calls

Consider an OT call between w_1 and w_2, where w_1 is the sender that inputs two bits (m^0, m^1), and w_2 is the receiver that inputs a bit b and receives the output bit m^b. In the compilation, the sender w_1 is emulated by two senders $p_{1,0}$ and $p_{1,1}$ that hold LRSS of m^0 and m^1, denoted $(\mathbf{m}_0^0, \mathbf{m}_1^0)$ and $(\mathbf{m}_0^1, \mathbf{m}_1^1)$, respectively. Similarly, the receiver w_2 is emulated by two receivers $p_{2,0}$ and $p_{2,1}$ that hold an LRSS of b, denoted $(\mathbf{b}_0, \mathbf{b}_1)$. The goal is to let the two receivers learn an LRSS of m^b and nothing else. To simplify notation, let us denote the two senders $(p_{1,0}, p_{1,1})$ by S_0 and S_1, and the two receivers $(p_{2,0}, p_{2,1})$ by R_0 and R_1. The challenge is to realize this 4-party protocol by using only calls to 2-party ideal functionalities.

We begin with a naive (and insecure) protocol, and then explain how to modify it to obtain a secure protocol. First, since the senders do not know the inputs of the receivers, they compute a secret sharing of the output for every possible input $(\mathbf{z}_0, \mathbf{z}_1)$ of the receivers. That is, for every pair of shares $(\mathbf{z}_0, \mathbf{z}_1)$ representing some bit z, the senders execute our two-party LRMPC for the functionality that recovers m^z and shares it into $(\mathsf{out}_0^{\mathbf{z}_0,\mathbf{z}_1}, \mathsf{out}_1^{\mathbf{z}_0,\mathbf{z}_1})$ where the first entry is delivered to S_0 and the second entry is deliverd to S^1. At this point, each party S_i holds a table of shares $(\mathsf{out}_i^{\mathbf{z}_0,\mathbf{z}_1})_{\mathbf{z}_0,\mathbf{z}_1}$ whose rows are indexed by all the possible inputs $\mathbf{z}_1$ of R_1, and its columns are indexed by all possible inputs $\mathbf{z}_0$ of R_0. Then, with the aid of two-party functionalities, S_0 lets R_0 learn all the $\mathbf{b}_0$'s column $(\mathsf{out}_0^{\mathbf{b}_0,\mathbf{z}_1})_{\mathbf{z}_1}$ and S_1 lets R_1 learn all the $\mathbf{b}_1$'s row $(\mathsf{out}_1^{\mathbf{z}_0,\mathbf{b}_1})_{\mathbf{z}_0}$. Observe that at the intersection of the row and the column is the sharing $(\mathsf{out}_0^{\mathbf{b}_0,\mathbf{b}_1}, \mathsf{out}_1^{\mathbf{b}_0,\mathbf{b}_1})$ of the correct output m^b of the receivers. Therefore, we can now let the receivers, that hold $(\mathbf{b}_0, \mathbf{b}_1)$, execute another two-party LRMPC to compute a fresh sharing of the output. In addition, for every $(\mathbf{z}_0, \mathbf{z}_1) \neq (\mathbf{b}_0, \mathbf{b}_1)$ the receivers hold at most one share from $(\mathsf{out}_0^{\mathbf{z}_0,\mathbf{z}_1}, \mathsf{out}_1^{\mathbf{z}_0,\mathbf{z}_1})$, so they have no information about m^{1-b}.

The naive protocol suffers from several problems. First, the senders' computation is exponential in the length ℓ_{LRSS} of the LRSS. Since we want an efficient solution with complexity polynomial in the security parameter κ, we must take ℓ_{LRSS} to be logarithmic in κ. This already means that the error of the LRSS, ϵ_{LRSS}, is only inverse polynomial in κ. In fact, the situation is much worse as the accumulated error is larger than 1. Indeed, the sender's computation is described by a circuit of size $s > 2^{\ell_{\mathsf{LRSS}}}$ so that the total error $s \cdot \epsilon_{\mathsf{LRSS}}$ is larger than 1.

To solve this problem we apply the LRSS with two different security parameters: we let $\sigma_0 = O(\log(\kappa))$ be the global LRSS security parameter that is used throughout the protocol, and we let $\sigma_1 = C \cdot \sigma_0$, for some constant $C > 1$, be the security parameter that is dedicated for the LRSS in the two-party LRMPC of the senders. In both cases, we employ the LRSS scheme of [SV19] that tolerates σ bits of leakage with share size of $\ell_{\mathsf{LRSS}} = O(\sigma)$ and an error of $\epsilon_{\mathsf{LRSS}} = 2^{-\Omega(\sigma)}$. By taking the constant C to be sufficiently large, we get a total error of $2^{O(\ell_{\mathsf{LRSS}}(\sigma_0))} \cdot \epsilon_{\mathsf{LRSS}}(\sigma_1) \ll 1$. In fact, the total error of the computation is now inverse polynomial in κ.

Another problem of the naive protocol is that it reveals m^0 and m^1 when a single sender and a single receiver are corrupted. Indeed, assume that the adversary corrupts S_0 and R_1, and recall that in this case we would like to think of the virtual players w_1 and w_2 as honest parties, so no information about m^0, m^1 and b should be revealed. The problem is that the sender S_0 and the receiver R_1 hold the shares $(\mathsf{out}_0^{\mathbf{z}_0,\mathbf{b}_1}, \mathsf{out}_1^{\mathbf{z}_0,\mathbf{b}_1})$ for every $\mathbf{z}_0$, and it is not hard to prove that they can recover both m^0 and m^1. We note that this problem arises since the senders hold LRSS of all the possible outputs. Therefore, to solve this problem, we prevent the senders from direct access to *any* share of the output by letting the senders execute LRMPC that computes *shares of shares*, i.e., instead of giving $\mathsf{out}_0^{\mathbf{z}_0,\mathbf{z}_1}$ to S_0 and $\mathsf{out}_0^{\mathbf{z}_0,\mathbf{z}_1}$ to S_1, we re-share those share among S_0 and S_1, so that S_j holds a single share of $\mathsf{out}_0^{\mathbf{z}_0,\mathbf{z}_1}$, denoted $\mathsf{out}_{0,j}^{\mathbf{z}_0,\mathbf{z}_1}$, and a single share of $\mathsf{out}_1^{\mathbf{z}_0,\mathbf{z}_1}$, denoted $\mathsf{out}_{1,j}^{\mathbf{z}_0,\mathbf{z}_1}$. The protocol now continues in a similar way to the naive protocol, where the senders transfer "the row" $(\mathsf{out}_{0,0}^{\mathbf{b}_0,\mathbf{z}_1}, \mathsf{out}_{0,1}^{\mathbf{b}_0,\mathbf{z}_1})_{\mathbf{z}_1}$ to R_0 and "the column" $(\mathsf{out}_{1,0}^{\mathbf{z}_0,\mathbf{b}_1}, \mathsf{out}_{1,1}^{\mathbf{z}_0,\mathbf{b}_1})_{\mathbf{z}_0}$ to R_1 with the aid of two-party ideal functionalities. Again, at the intersection of the row and column the receivers hold the shares $((\mathsf{out}_{0,0}^{\mathbf{b}_0,\mathbf{b}_1}, \mathsf{out}_{0,1}^{\mathbf{b}_0,\mathbf{b}_1}), (\mathsf{out}_{1,0}^{\mathbf{b}_0,\mathbf{b}_1}, \mathsf{out}_{1,1}^{\mathbf{b}_0,\mathbf{b}_1}))$ that fully define $(\mathsf{out}_0^{\mathbf{b}_0,\mathbf{b}_1}, \mathsf{out}_1^{\mathbf{b}_0,\mathbf{b}_1})$ that shares m^b, and can therefore apply two-party LRMPC to obtain fresh shares of the output. In Section 5 we prove that this protocol is indeed leakage resilient.

2.1.4 Pairwise Correlations

We move on and explain how to replace the calls to two-party ideal functionalities by using correlated randomness. The high level idea is to replace every call to an ideal functionality $\mathcal{F}$ with a two-party leakage-resilient secure computation in the correlated randomness model. To do so, we present a simple two-party protocol in the correlated randomness model, that can be seen as a special case of the protocol of [SV19]. The complexity of our protocol is *exponential* in the circuit-depth of $\mathcal{F}$. However, all the ideal functionalities that we use have depth logarithmic in the security parameter (in fact, even logarithmic input length—in this sense they are "simple"), and therefore the overall complexity is polynomial in the security parameter. Full details of this transformation appear in Section 6.

2.1.5 Threshold LRMPC

So far we have considered the case of passive security with optimal resiliency of $n-1$. We continue with the following non-standard *threshold* version of LRMPC. Given a threshold t we require the following properties:

- *Leakage-resilient passive security* against an adversary that corrupts a set Z of at most $t-1$ parties (and receives leakage from the honest parties).
- *Perfect correctness with abort* for t honest parties: for any set S of at least t honest parties, and any active (aka malicious) adversary that corrupts the remaining $n-t$ parties and may

deviate from the protocol, the output of the honest parties is either an abort, or in the image of the function f, restricted to the inputs of the honest parties. Furthermore, in the latter case a corresponding pre-image can be extracted efficiently from the views of the (honest) parties in S.

Note that our LRMPC protocol is a threshold LRMPC with threshold $t = n$, as perfect correctness with abort is trivially satisfied in this case. Threshold LRMPC can be generalized to χ-LRMPC, where χ is a monotone predicate. In this case, the leakage-resilient passive security requirement should hold with respect to *unauthorized* sets Z with $\chi(Z) = 0$, and the perfect-correctness-with-abort requirement should hold with respect to *authorized* sets S with $\chi(S) = 1$. Taking χ to be the t-threshold function Th_t^n we get the notion of t-threshold LRMPC.

The work of [AK25] constructed threshold MPC in the OT-hybrid model (without leakage-resilience) for any $t \leq n$, and showed a generic way to transform it into NPSS. In fact, their construction applies to any monotone formula χ of logarithmic depth, i.e., for every such formula χ they constructed χ-MPC in the OT-hybrid model.[3] In this paper, we construct threshold LRMPC for any $2 \leq t \leq n$, and show that essentially the same transformation implies leakage-resilient NPSS. We proceed with an overview of the construction.

Let us fix any $2 \leq t \leq n$. Our starting point is a logarithmic-depth formula χ for the threshold function Th_t^n, that satisfies the following special property: All the input gates of the formula (i.e., gates whose incoming wires are input wires) are AND gate with fan-in 2. In Claim 7.4 we show an explicit construction of such a formula with logarithmic depth. Letting m denote the number of AND gates at the bottom (input) level of the formula χ, we can decompose χ into a triple (h, AND_2^m, T) as follows: (1) $h : \{0,1\}^m \to \{0,1\}$ is the sub-formula of χ that consists of all the gates of χ except the input gates, and has m input wires; (2) $\mathsf{AND}_2^m : \{0,1\}^{2m} \to \{0,1\}^m$ consists of m parallel 2-ary AND gates (the bottom layer of χ); and $T : [m] \times [2] \to [n]$ is a mapping that maps an input wire into an input variable. That is, we identify each input wire by a pair $(i, b) \in [m] \times [2]$ where $i \in [m]$ specifies an input gate and $b \in [2]$ specifies whether it is the left or right input wire of the i-th input gate.

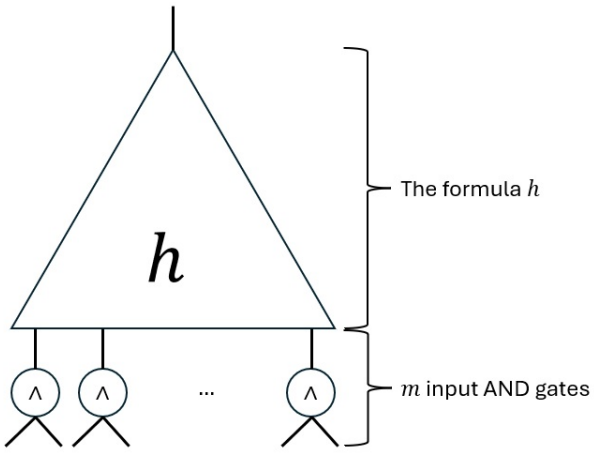

Figure 1: Decomposition of χ

[3] The restriction to logarithmic depth is actually redundant since any monotone formula of polynomial-size can be converted into an equivalent formula of logarithmic depth via standard balancing techniques [Spi71, Weg83].

First, we use the results of [AK25] to obtain an h-MPC protocol π' (that does not satisfy leakage resilience), and we note that π' involves m parties $w_1, \ldots, w_m$ that correspond to the input wires of h. Just like in our original LRMPC protocol, we construct an n-party protocol π from π' by letting some pair of parties p_i and p_j emulate every w_k by holding a LRSS of w_k's view. The mapping is done according to the map T, i.e., the parties that emulate w_k are set to be $p_{T(k,1)}$ and $p_{T(k,2)}$, and the emulation is done just like in our original protocol.

To see that leakage-resilient passive security holds, we recall that the view of w_i is revealed to the adversary only if both $p_{T(i,1)}$ and $p_{T(i,2)}$ are corrupted. Therefore, for any set $Z \subseteq \{p_1, \ldots, p_n\}$ of $t-1$ corrupt parties, we define the corresponding set of corrupt virtual players w_i to be $Z_w = \{w_i : p_{T(i,1)}, p_{T(i,2)} \in Z\}$. The main observation now is that $0 = \chi(Z) = h(Z_w)$, and therefore privacy follows by the privacy of the h-MPC π'. As for active correctness with abort, we note that even if one of $p_{T(i,1)}$ and $p_{T(i,2)}$ is corrupted by the malicious adversary, then the virtual player w_i behaves maliciously. Therefore, for any set of t honest parties S, the corresponding honest virtual players w_i are $S_w = \{w_i : p_{T(i,1)}, p_{T(i,2)} \in S\}$. Again, we use the fact that $1 = \chi(S) = h(S_w)$, to argue that the perfect correctness with abort of π' implies perfect correctness with abort in π. Full details of the construction appear in Section 5.

2.2 NIZK Amplification

Let us get back to the problem of NIZK amplification. For simplicity, we focus on amplifying an (α, β)-weak NIZK Π_{wk} with *statistical* soundness, where $\alpha(\kappa) + \beta(\kappa) < 1 - 1/\kappa$. The amplification consists of two steps. In the first step, which is more technically involved, we transform Π_{wk} into a scheme Π with negligible statistical soundness error and inverse-polynomial zero-knowledge error, of say $1/\kappa$. In the second step, we further reduce the zero-knowledge error to negligible. Before presenting these amplifiers, let us first get a better understanding of the notion of β-zero-knowledge.

On β-zero-knowledge. Recall that $\beta(\kappa)$-zero-knowledge guarantees that for every circuit-SAT instance f and satisfying assignment $\mathbf{x}$, every polynomial-time adversary has advantage at most $\beta(\kappa)$ in distinguishing between a "real" CRS/proof pair (crs, π) from a simulated pair that is generated by $\mathsf{Sim}_{wk}(1^\kappa, f)$. A basic component of our analysis is the indistinguishability hardcore lemma of [MT10] that, loosely speaking, implies that for every polynomial $s(\kappa)$ there is a real-world event A and a simulation event B, both occurring with probability $1 - \beta(\kappa)$, such that every size-$s(\kappa)$ adversary has only a negligible advantage in distinguishing a real CRS/proof pair, (crs, π), conditioned on the event A occurring, from a simulated CRS/proof pair, $\mathsf{Sim}_{wk}(1^\kappa, f)$, conditioned on the event B occurring. We emphasize that the events A and B depend on f and $\mathbf{x}$. Since the events imply indistinguishability, we think of them as the "good" events. From now on, to simplify the discussion, let us focus on size-$s(\kappa)$ distinguishers for some concrete polynomial $s(\kappa)$.

2.2.1 The First Amplifier

Naive attempt: t-out-of-n NPSS. Observe that if we prove n statements, for a sufficiently large n and using independent CRS's, then with overwhelming probability the good events fail to occur in less than $(\beta(\kappa) + \frac{1}{10\kappa})n$ of the proofs. Therefore, our first try is to amplify using a t-out-of-n NPSS for $t = (\beta(\kappa) + \frac{1}{10\kappa})n$ and a sufficiently large n. That is, we let the prover use the instance

mapper of the NPSS on f to obtain n instances $f_1, \ldots, f_n$, and the assignment mapper on $(f, \mathbf{x})$ to obtain the global assignment $\mathbf{y}$ together with the partial assignments $\mathbf{y}_1, \ldots, \mathbf{y}_n$. The prover then commits to every bit of the global assignment $\mathbf{y}$, and the new proof consists of the vector of commitments $\mathbf{C}^{\mathbf{y}}$, together with n sub-proofs $\pi_1, \ldots, \pi_n$, where the i-th sub-proof π_i proves the following "extended" statement:

> **Statement $\hat{f}_i$:** There is a partial assignment $\mathbf{y}_i$ that satisfies f_i and is consistent with the commitments $\mathbf{C}^{\mathbf{y}}$.

Amplifying statistical soundness. First, let us observe that statistical soundness holds. Indeed, $(1-\alpha(\kappa))n \geq (\beta(\kappa)+\frac{1}{\kappa})n = t + \frac{9}{10\kappa}n$, so if all the sub-proofs are accepted by the verifier, then with overwhelming probability at least $t+1$ of them are correct and we let $\mathcal{Y} := (\mathbf{y}_{i_1}, \ldots, \mathbf{y}_{i_{t+1}})$ be the corresponding partial assignments that satisfy $f_{i_1}, \ldots, f_{i_{t+1}}$. If, in addition, the underlying non-interactive commitment is statistically-binding, then with overwhelming probability the partial assignments in $\mathcal{Y}$ are pairwise consistent on Boolean values. Therefore, by the recovery property of the NPSS, the original circuit f is satisfiable.

Simulation attempt. From now let us focus on zero-knowledge amplification. To make things easier, we only strive for inefficient simulation also known as *witness indistinguishability*. By using standard techniques [FLS90], we can later upgrade this into an efficient simulator with a small loss in the parameters. Recall that with overwhelming probability the good event fails to occur in at most $t-1$ of the sub-proofs π_i. Therefore, we focus on the task of simulating a proof, given that the good event failed to occur in some subset $Z \subseteq [n]$ of size at most $t-1$ of the sub-proofs. We denote by $I = [n] \setminus Z$ the set of all indices where the good event occurred, and by A_i and B_i the good events for the i-th sub-proof π.

We begin by simulating the the proofs for the "bad instances" $i \in Z$. The privacy of the NPSS implies that the $t-1$ partial assignments $(\mathbf{y}_i)_{i \in Z}$ can be perfectly simulated, and therefore the corresponding proofs $(\pi_i)_{i \in Z}$ conditioned on the good events not occurring can be perfectly simulated as well.[4]

We move on to handle good indices. Fix some good index $i \in I$ and consider a proof π_i. For such an index, the simulator does not hold the corresponding partial assignment $\mathbf{y}_i$. Since the good event A_i happens, we would like to apply Sim_{wk} on $\hat{f}_i$ conditioned on B_i. However, the event $B_i = B_i(\mathbf{y}_i)$ depends on the partial assignment $\mathbf{y}_i$ that is not known to us. Therefore it is not clear that we can sample (even inefficiently) from the conditional distribution without knowing $\mathbf{y}_i$.

Solution: leakage-resilient NPSS. To overcome this difficulty we follow an idea of [GP99] that translates a simulator that works well when conditioned on some large sub-space (the event $B_i(\mathbf{y}_i)$ in our case) into a simulator that works well over the entire probability space given some short "hint" or leakage from the partial assignment $\mathbf{y}_i$. The idea is to use the leakage function in order to find a good random string for the simulator Sim_{wk} under which $B_i(\mathbf{y}_i)$ happens. Specifically, we sample a vector $\mathbf{r}^i = (r^i_1, \ldots, r^i_\ell)$ that contains ℓ possible internal random strings for the simulator Sim_{wk}, and define the leakage function $\tau_i^{\mathbf{r}^i}$, that takes as an input the partial assignment $\mathbf{y}_i$ and

[4] The simulation may be inefficient since the conditional distribution is not necessarily efficiently samplable. Also, in order to generate the sub-proofs $(\pi_i)_{i \in Z}$, we need to somehow generate the commitments— let us get back to this point later on.

outputs the first index $j \in [\ell]$ for which the event $B_i(\mathbf{y}_i)$ occurs if the randomness of Sim_{wk} is r^i_j. By taking $\ell = \ell(\kappa)$ to be a sufficiently large polynomial, we can guarantee that, with overwhelming probability over the choice of $\mathbf{r}^i$, such an index j exists, and we can sample from the conditional distribution $B_i(\mathbf{y}_i)$. Overall, this means that simulation can be done given a leakage of $O(\log(\kappa))$ bits from $\mathbf{y}_i$.

Now, if we replace the t-out-of-n NPSS with a t-out-of-n *leakage-resilient* NPSS, we can simulate (up to a small statistical distance) the shares $(\mathbf{y}_i)_{i\in Z}$ together with the leakage $(\mathsf{leak}_i := \tau_i^{\mathbf{r}^i}(\mathbf{y}_i))_{i\in I}$, where $(\mathbf{r}^i)_{i\in I}$ are independent random vectors. Like before, for $i \in Z$ we can use the partial assignment $\mathbf{y}_i$ to sample the proof π_i. For $i \in I$, if we parse leak_i as an index j_i, then we know that if we execute $\mathsf{Sim}_{wk}(1^\kappa, \hat{f}_i; \mathbf{r}^i[j_i])$ then it is equivalent to sampling $\mathsf{Sim}_{wk}(1^\kappa, \hat{f}_i)$ conditioned on B_i occurring.

On trapdoor commitments. So far we have ignored the fact that the extended statements $\hat{f}_1, \ldots, \hat{f}_n$ and the proofs $\pi_1, \ldots, \pi_n$ depend on the commitments $\mathbf{C}^y$. Naively, we could try to simulate the commitments in the following way: given the simulated partial assignments $(\mathbf{y}_i)_{i\in Z}$, generate a fake global assignment $\mathbf{y}'$ that is consistent with $(\mathbf{y}_i)_{i\in Z}$ and is 0 in every entry that is not defined by $(\mathbf{y}_i)_{i\in Z}$, and commit to $\mathbf{y}'$.

Since the simulator is inefficient, it is crucial that the commitment scheme is *statistically hiding*. Indeed, the proofs $(\pi_i)_{i\in Z}$ that depend on $\mathbf{C}^y$ are sampled inefficiently by the simulator. In addition, the inefficient leakage functions $\tau_i^{\mathbf{r}^i}$ depend on the extended statement $\hat{f}_i$ and therefore on the commitments $\mathbf{C}^y$. So in order to preserve indistinguishability, we must employ statistically-hiding commitments. However, such commitments offer only computational binding and so soundness will be degraded to be computational. (We will also need to rely on stronger assumptions since statistically hiding commitments are not known to follow from one-way functions.)

To solve this mismatch, we use *trapdoor commitments* [FS89]. Roughly, trapdoor commitments are statistically binding and computationally hiding. In addition, they provide a "fake" mode of operation, where the public parameters of the commitment scheme are generated together with a trapdoor. This mode of operation is indistinguishable from the "honest" mode of operation, and we can use the trapdoor to obtain perfectly hiding commitments (but without the binding property). Our NIZK amplifier uses the honest mode of operation, and therefore provides statistical soundness. In addition, in the zero-knowledge analysis where the prover is already assumed to be honest and therefore we don't care about the binding property, we move to the (indistinguishable) fake mode of operation that provides perfect hiding.

2.2.2 The Second Amplifier: Reducing the ZK Error to Negligible

Our first amplifier inherits its ZK error from the privacy error of the LRNPSS, and so it is only inverse polynomial. To further reduce the error to negligible we employ the (non-leakage resilient) NPSS from [AK25] that servers as a NIZK-combiner, and show that in the low-error regime a combiner can be used as an amplifier. This phenomena was previously noticed in other settings (e.g., [Tes10, LM20]) and was recently adopted to the setting of statistical zero-knowledge by [BKP+24]. We extend these results to the computational setting via the use of a new variant of the XOR lemma. (See Sections 8.1.2 and 8.3.)

3 MPC: Basic Definitions

In the following section, we define MPC and present our non-standard security definitions. Since we will later manipulate protocols (via different forms of compositions), we will have to carefully define the syntax of a "protocol". This can be done in multiple ways, and our concrete formalization follows the framework of Hirt and Maurer [HM00].

3.1 Formalizing Protocols

Functionality. An n-party (deterministic) functionality $\mathcal{F}$ is a function that takes n inputs $\mathbf{x}_1, \ldots, \mathbf{x}_n$ and provides n outputs $\mathbf{y}_1, \ldots, \mathbf{y}_n$. We think of $\mathbf{x}_i$ as the input of the i-th party, and of $\mathbf{y}_i$ as the output of the i-th party.

Variable space, view and abort flag. We assume that the parties communicate over private point-to-point channels and use a broadcast channel to announce an "abort". For concreteness, we formalize the syntax of a protocol via the framework of [HM00, CDI+13]. We assume a global *variable space* $\mathcal{X}$. A variable $x \in X$ can take values from the binary field $\mathbb{F}_2$. Every value generated during a protocol execution, including inputs, local data and outputs, is assigned to a variable. We assume that every variable is assigned a value exactly once, and can't be re-written. In addition, every variable belongs only to a single party, and we let the *view* of a party p, denoted $\mathsf{view}(p)$, be the values of all the variables that belong to p. For a variable $x \in \mathcal{X}$, we denote the value assigned to x by $\mathsf{val}(x)$.

We also assume the existence of a public *abort flag*, denoted flag, that all the parties can access. The value of the flag is set to zero at the beginning of the execution, and at any time each party can raise the flag (i.e., set its value to 1). Once the value of the abort flag is set to 1, it cannot be changed. Jumping ahead, at the end of the protocol the parties output "abort" if and only if the flag is turned on.[5]

Protocols. A protocol π among a set $\mathcal{P}$ of parties over the variable space $\mathcal{X}$ is a sequence $d_1, \ldots, d_L$ of *statements*. There are several types of statements:

- An *input statement* $\mathsf{input}(p, i, x)$ instructs the party $p \in \mathcal{C}$ to read the i-th bit from its input tape and to assign the value to the variable $x \in \mathcal{X}$ that belongs to p.
- An *output statement* $\mathsf{out}(p, i, x)$ instructs the party $p \in \mathcal{C}$ to set its i-th output bit to be $\mathsf{val}(x)$ where x belongs to p. However, if the public abort flag is raised, i.e., $\mathsf{flag} = 1$, then the output is set to be $\perp$.
- A *transmit statement* $\mathsf{transmit}(p_1, p_2, x_1, x_2)$ instructs a party $p_1 \in \mathcal{P}$ to send the value of the variable x_1 that belongs to p_1 to a party $p_2 \in \mathcal{P}$, and it instructs p_2 to assign the received value to the variable x_2.
- A *computation statement* $\mathsf{comp}(p, \mathsf{op}, X, x)$ instructs a party $p \in \mathcal{P}$ to perform the atomic operation op on the tuple of variables $X \subseteq \mathcal{X}$ that are in its view and to assign the result to

[5] One could emulate such a flag using a broadcast channel. However, for technical reasons it will be convenient to use the flag mechanism and avoid the use of a broadcast channel.

the variable $x \in \mathcal{X}$. For concreteness, we assume that $\mathsf{op} \in \{+, \times, \mathsf{rand}\}$ where $\{+, \times\}$ are addition and multiplication over $\mathbb{F}_2$ (in which case X contains exactly two variables), and rand is an instruction to assign a random value to x (in which case $X = \emptyset$).

- An *abort statement* $\mathsf{abort}(p, x)$ instructs a party $p \in \mathcal{P}$ to raise the abort flag flag if the value of the variable x in the view of p is 1.

Let $\mathcal{G}$ be a 2-party functionality, that takes an input $\alpha \in \{0,1\}^{\ell_1}$ from the first party, an input $\beta \in \{0,1\}^{\ell_2}$ from the second party, computes $\gamma = \mathcal{G}(\alpha, \beta)$, where $\gamma \in \{0,1\}^{\ell_3}$, and returns γ only to the second party (the first party has no output). We call such functionalities two-party *solitary-output* functionalities. A protocol π in the $\mathcal{G}$-hybrid model gives every pair of parties access to the functionality $\mathcal{G}$, and allows the following statement:

- A *$\mathcal{G}$-statement* $\mathsf{func}(\mathcal{G}, p_1, p_2, X^1, X^2, X^3)$ for parties $p_1, p_2 \in \mathcal{P}$, a tuple $X^1 = (x^1_1, \ldots, x^1_{\ell_1})$ of variables that belong to p_1, a tuple $X^2 = (x^2_1, \ldots, x^2_{\ell_2})$ of variables that belong to p_2, and a tuple $X^3 = (x^3_1, \ldots, x^3_{\ell_3})$ of variables that belong to p_2, instructs the parties to use the functionality $\mathcal{G}$ in the following way. Party p_1 inputs $\alpha \in \{0,1\}^{\ell_1}$ defined by the values of the variables in X^1, and p_2 inputs $\beta \in \{0,1\}^{\ell_2}$ defined by the values of the variables in X^2. The output from the functionality $\gamma = \mathcal{G}(\alpha, \beta) \in \{0,1\}^{\ell_3}$ is then given by the functionality to p_2 that assigns γ to the variables in X^3.

Throughout, we only consider protocols π that are well defined, i.e., variables are only accessed after they were defined, and the protocol does not attempt to re-write a variable. In addition we assume that all the input statement appear first as a prefix of the protocol, and that every input bit is read at most once; we assume that all the output statements appear at the end as a suffix of π, and that every output bit is written at most once; and we assume that those are the only input/output statements in the protocol. Finally, we assume that for every choice of inputs to the parties, in an honest execution of π the abort flag is never raised.

3.2 Security Definition

From now on, let $\mathcal{F}$ be an n-party functionality, let π be a protocol over the set of parties $\mathcal{P}$.

Our security definition is parameterized by a monotone predicate $\chi : 2^{\mathcal{P}} \to \{0,1\}$ referred to as an *access structure*. Here monotonicity means that $\chi(A) \leq \chi(B)$ for every $A, B \subseteq \mathcal{P}$ that satisfy $A \subseteq B$. Our security definition consists of two parts. The first part requires standard perfect privacy and perfect correctness against a passive adversary that corrupts subsets Z for which $\chi(Z) = 0$. (Intuitively, as in the case of secret sharing these coalitions correspond to "unauthorized coalitions".)

Definition 3.1 (χ-passive security)**.** *We say that π computes $\mathcal{F}$ with* χ-passive security *if there exists a simulator* Sim *such that for all inputs $(x_1, \ldots, x_n)$ to the parties the following holds:*

- (Perfect Correctness) *With probability 1, the output of the i-th party in an execution of π is y_i, where $(y_1, \ldots, y_n) = \mathcal{F}(x_1, \ldots, x_n)$.*
- (Perfect Privacy) *For any set $Z \subseteq \mathcal{P}$ of corrupt parties that satisfies $\chi(Z) = 0$ it holds that*

$$(\mathsf{view}(p))_{p \in Z} \equiv \mathsf{Sim}(Z, (x_i)_{c_i \in Z \cap \mathcal{C}}, (y_i)_{c_i \in Z \cap \mathcal{C}})$$

where $(y_1, \ldots, y_n) = \mathcal{F}(x_1, \ldots, x_n)$. In addition, the running time of Sim *is polynomial in n and the number of statements in π.*

The second part of the definition guarantees *perfect correctness* for authorized coalitions S for which $\chi(S) = 1$, even in the presence of an active adversary corrupts all other parties $Z := \mathcal{P} \setminus S$. That is, the honest parties in S will *never* generate an erroneous output but the adversary may force them to output an "abort" symbol. This means that in every execution of π in the presence of the active adversary, where the parties hold inputs $x = (x_1, \ldots, x_n)$, either the abort flag is raised, or the outputs of the honest parties are consistent with $\mathcal{F}(x'_1, \ldots, x'_n)$ for some assignment $x' = (x'_1, \ldots, x'_n)$ that is consistent with x with respect to the honest parties (i.e., $x'_i = x_i$ if the i-th party is honest). In fact, we even require the existence of an efficient extractor so that whenever the honest parties do not abort, the assignment $(x'_1, \ldots, x'_n)$ can be extracted from the views of the honest parties. Before formalizing this non-standard notion, let us formally define the notion of active adversary in our concrete syntactic framework.

Active adversarial behavior. As usual, an active adversary that corrupts a set Z of parties is an adversary that can make the corrupted parties in Z deviate from the protocol. Since our non-standard notion of correctness should hold with probability 1, it is enough to require that it holds with respect to *degenerate active adversaries*, that specify *before* the execution of π what messages the corrupt parties send to the honest parties. Indeed, an active adversary that violates correctness with positive probability can be transformed into a degenerate active adversary that violates correctness with positive probability. Since correctness is satisfied when the abort flag is raised, we may also assume that the adversary never raises the abort flag. Finally, we mention that we allow the active adversary more power: for every $\mathcal{G}$-statement where the first party is corrupt and the second party that receives the output is honest, we let the adversary *choose* the output of the functionality $\mathcal{G}$.

Formally, let $\pi = (d_1, \ldots, d_L)$ be a protocol over $\mathcal{P}$. An *active adversarial behavior* with respect to a set Z of corrupt parties, is a protocol $\tilde{\pi} = (\tilde{d_1}, \ldots, \tilde{d_L})$ such that for every $i \in [L]$, (1) if the i-th statement is a transmit statement $\mathsf{transmit}(p_1, p_2, x_1, x_2)$ where $p_1 \in Z$ and p_2 is honest, then the statement is marked by a message m, so that in an execution of $\tilde{\pi}$ party p_1 ignores the original statement and instead sends m to p_2, (2) if the i-th statement is a $\mathcal{G}$-statement $\mathsf{func}(\mathcal{G}, p_1, p_2, X^1, X^2, X^3)$ where p_1 is corrupt and p_2 is honest, then the statement is marked by an output γ, so that in an execution of $\tilde{\pi}$ the output of $\mathcal{G}$ is always γ, (3) if the i-th statement is $\mathsf{abort}(p, x)$ for $p \in Z$, then the statement is ignored, and (4) otherwise, $\tilde{d_i} = d_i$ is executed honestly.

Definition 3.2 (χ-perfect active correctness with abort)**.** *We say that π computes $\mathcal{F}$ with* χ-perfect active correctness with abort*, if there exists a deterministic extractor* Ext *such that for every set of $S \subseteq \mathcal{P}$ of honest parties that satisfies $\chi(S) = 1$, all active adversarial behaviors $\tilde{\pi}$ with respect to the corrupt parties $Z := \mathcal{P} \setminus S$, all inputs $(x_1, \ldots, x_n)$ to the parties, and every execution of $\tilde{\pi}$, one of the following holds:*

- *The output of all honest parties is $\perp$.*
- *There exists a vector of inputs $\mathbf{x}^* = (x_1^*, \ldots, x_n^*)$ such that (1) $x_i^* = x_i$ for all honest parties $p_i \in \mathcal{P}$, (2) the outputs of the honest parties are consistent with $\mathcal{F}(\mathbf{x}^*)$, and (3)* Ext*, given the set Z and the view of all honest parties, outputs $\mathbf{x}^*$.*

In addition, the running time of Ext *is polynomial in n and the number of statements in π.*

Our default notion of security, denoted by χ-dual security, requires privacy for authorized sets and correctness for unauthorized sets.

Definition 3.3 (χ-dual security)**.** *We say that π computes $\mathcal{F}$ with* χ-dual security*, if π provides both χ-passive security and χ-perfect active correctness with abort.*

4 Leakage Resilient MPC: Basic Definitions

In this section we present basic definitions of leakage-resilient MPC. At a high level, in leakage-resilient MPC a small amount of information is leaked from every honest party. That is, we assume that for every p_i there is a leakage function μ_i that takes as an input the view of p_i and outputs μ bits. The function μ_i can be computationally unbounded but has to be selected non-adaptively before the execution of the protocol. We follow the definition of [BDIR21, Section 5], and consider the task of evaluating some circuit f on an input $\mathbf{x}$, by first encoding the input $\mathbf{x}$ to a tuple $(\mathsf{input}_1, \ldots, \mathsf{input}_n)$, and then executing the protocol on the encoded input, where p_i holds input_i as an input. The goal is to let all the parties learn $f(\mathbf{x})$ when the protocol terminates. This is formalized in the following way.

Definition 4.1 (Protocols with encoded input)**.** *An n-party protocol for a circuit $f : \{0,1\}^\ell \to \{0,1\}^m$ with encoded input is a tuple $\Pi = (\pi, \mathsf{Enc}, \mathsf{Rand})$ where π is a protocol (i.e., a list of statements) among $\mathcal{P}$,* Enc *is a randomized input encoder and* Rand *is a correlated-randomness generator. An execution of Π on input $\mathbf{x} \in \{0,1\}^\ell$ is done by encoding the input $(\mathbf{x}_1, \ldots, \mathbf{x}_n) \leftarrow \mathsf{Enc}(\mathbf{x})$, sampling correlated randomness $(\mathbf{r}_1, \ldots, \mathbf{r}_n) \leftarrow \mathsf{Rand}$ and then executing the protocol π, where the i-th party holds as an input the string $(\mathbf{x}_i, \mathbf{r}_i)$. We mention that in some cases we consider protocols that require no correlated randomness, and that in this case we simply omit* Rand *from the tuple.*

We say that the protocol has *pairwise correlations*, if there are randomized algorithms $(\mathsf{Rand}_{i,j})_{1 \le i < j \le n}$, such that Rand samples $(\mathbf{r}^i_{i,j}, \mathbf{r}^j_{i,j}) \leftarrow \mathsf{Rand}_{i,j}$, and sets the randomness of the i-th party to be $(\mathbf{r}^i_{i,j})_{j>i}, (\mathbf{r}^i_{j,i})_{j<i}$.

4.1 Security Definition

Throughout the section we use the notation $A \equiv B$ to indicate that the random variables A and B are identically distributed and $A \equiv_\epsilon B$ to indicate that the underlying probability distributions are ϵ-close in statistical distance. (See Section A.1 for basic definitions and notations.)

In the following we take $\mathcal{P} = \{p_1, \ldots, p_n\}$ to be a set of n parties, and we let $f : \{0,1\}^\ell \to \{0,1\}^m$ be a circuit. We continue with the security definition of leakage resilient MPC. Let $\chi : 2^{\mathcal{P}} \to \{0,1\}$ be a monotone predicate.

Definition 4.2 (Leakage resilient passive security)**.** *For a positive integer μ and for $\epsilon > 0$, we say that $\Pi = (\pi, \mathsf{Enc}, \mathsf{Rand})$ computes f with (χ, μ, ϵ)-*leakage resilient passive security *if there exists a simulator* Sim *such that for every input $\mathbf{x} \in \{0,1\}^\ell$ and for all functions $\tau_1, \ldots, \tau_n$, where τ_i takes the view of the i-th party and outputs μ bits, the following holds:*

- (Perfect Correctness) *With probability* 1 *over the randomness of π,* Enc *and* Rand*, when the parties execute Π on input $\mathbf{x}$, the output of all parties is $f(\mathbf{x})$.*
- (Statistical Privacy) *For any set $Z \subseteq \mathcal{P}$ of parties such that $\chi(Z) = 0$,*

$$((\mathsf{view}(p))_{p \in Z}, (\tau_i(\mathsf{view}(p)))_{p \notin Z}) \equiv_\epsilon ((\mathsf{view}'(p))_{p \in Z}, (\tau_i(\mathsf{view}'(p)))_{p \notin Z}),$$

where $\mathsf{view}(p)$ *is the view of the party* p *in an execution of* Π *on input* $\mathbf{x}$*, and* $(\mathsf{view}'(p_1), \ldots, \mathsf{view}'(p_n)) \leftarrow \mathsf{Sim}(Z, f(\mathbf{x}))$.

The above definition is non-asymptotic and treats the parameters n, μ and ϵ as finite. Jumping ahead, our asymptotic constructions will be obtained by presenting efficient compilers that map (f, χ, μ, ϵ) to (Π, Sim).

We have a special interest in the case where χ is a threshold function, i.e., there is some integer $t \leq n$ such that χ returns 1 on any string of weight at least t, and zero otherwise. In this case, we may say that Π is a t-out-of-n protocol that computes f with (μ, ϵ)-leakage-resilient passive security.

We mention that some of our protocol involve no correlated randomness, and in this case we simply omit Rand from the tuple Π. The following definition of perfect active correctness with abort, will only be used on such protocols with no correlated randomness.

Definition 4.3 (Perfect active correctness with abort)**.** *We say that* $\Pi = (\pi, \mathsf{Enc})$ *computes* f *with* χ-perfect active correctness with abort*, if there exists a deterministic extractor* Ext *such that for every set* $S \subseteq \mathcal{P}$ *of honest parties that satisfies* $\chi(S) = 1$*, for* $Z := \mathcal{P} \setminus S$*, for all adversarial behaviors* $\tilde{\pi}$ *with respect to* Z*, every encoding* $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ *(even if it is not in the support of* Enc*), and every execution of* $\tilde{\pi}$ *with inputs* $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$*, one of the following holds:*

- *The output of all honest parties is* $\perp$*.*
- *The outputs of the honest parties is a bit* $b \in \{0, 1\}$ *and* Ext*, given the set* S *and the views of all the parties in* S*, outputs an input* $\mathbf{x}$ *with* $f(\mathbf{x}) = b$*.*

Leakage-resilient dual security. We say that $\Pi = (\pi, \mathsf{Enc})$ computes f with (χ, μ, ϵ)*-leakage-resilient dual security*, if Π provides both (χ, μ, ϵ)-passive security and χ-perfect active correctness with abort.

Remark 4.4 (On the definition of prefect active correctness with abort.)**.** *While our definition of perfect active correctness with abort suffices for the construction of leakage-resilient NPSS from leakage-resilient MPC, our construction of leakage-resilient MPC will satisfy even stronger properties: In our construction,* every *encoding* $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ *is in the support of* Enc*, and for every subset* S *of honest parties that satisfies* $\chi(S) = 1$*, the extractor, given the views of the honest parties, outputs the* unique *input* $\mathbf{x}$ *that is consistent with the encoding of the honest parties.*

4.2 The Secret Sharing Scheme

In leakage-resilient secret sharing, the privacy of the secret is preserved against any unauthorized set, even when a small amount of information is leaked from every share that belongs to a party outside the set. As we will only be interested in 2-out-of-2 leakage-resilient secret sharing, we continue with a formal definition for this special case.

Definition 4.5. *Let* μ *be a positive integer and* $\epsilon > 0$*. A 2-party* (μ, ϵ)*-local-leakage-resilient secret sharing is a pair of efficient algorithms* $(\mathsf{Share}, \mathsf{Recover})$ *with the following syntax:*

- Share *is a randomized algorithm that takes as an input a secret bit* $s \in \{0, 1\}$*, and outputs two shares* $(\mathbf{s}_0, \mathbf{s}_1)$*.*

- $\mathsf{Recover}$ *is a deterministic algorithm that takes as an input two shares* $(\mathbf{s}_0, \mathbf{s}_1)$ *and outputs* $s \in \{0, 1, \perp\}$.

The algorithms are required to satisfy the following properties:

- (Correctness) *For every* $s \in \{0, 1\}$,

$$\Pr[\mathsf{Recover}(\mathsf{Share}(s)) = s] = 1.$$

- (Privacy) *For every functions* τ_0, τ_1 *that take as an input a share and output* μ *bits, and for every set* $Z \subseteq \{0, 1\}$ *of size at most* 1 *it holds that*

$$(\mathbf{s}_i)_{i \in Z}, (\tau_i(\mathbf{s}_i))_{i \notin Z} \equiv_\epsilon (\mathbf{s}'_i)_{i \in Z}, (\tau_i(\mathbf{s}'_i))_{i \notin Z},$$

where $(\mathbf{s}_0, \mathbf{s}_1) \leftarrow \mathsf{Share}(0)$ *and* $(\mathbf{s}'_0, \mathbf{s}'_1) \leftarrow \mathsf{Share}(1)$.

Next, we describe our leakage-resilient secret sharing scheme, which is a special case of the scheme of [SV19]. As part of the construction we use average-case strong extractor, and the reader is referred to Section A.1 for a formal definition.

Construction 4.6. *Let* $(\mathsf{Share}_\ell, \mathsf{Recover}_\ell)$ *be an additive 2-out-of-2 secret sharing scheme for an* ℓ*-bit secret, i.e.,* $\mathsf{Share}_\ell(s)$ *samples a random* ℓ*-bit string* r*, and defines the shares* $s_1 = r, s_2 = s \oplus r$ *and* $\mathsf{Recover}_\ell(s_1, s_2) = s_1 \oplus s_2$*. Let* $\mathsf{acExt} : \{0,1\}^a \times \{0,1\}^b \to \{0,1\}$ *be an average-case* $(a - \mu, \epsilon)$*-strong extractor. We define* $\mathsf{LRSS} = (\mathsf{LRShare}, \mathsf{LRRecover})$ *in the following way.*

- ($\mathsf{LRShare}$) *To share a secret* $s \in \{0, 1\}$*, sample a 2-out-of-2 sharing of* s*,* $(s'_1, s'_2) \leftarrow \mathsf{Share}_2(s)$*, together with a random a seed* $\sigma \leftarrow \{0,1\}^b$ *and a masking bit* $r \leftarrow \{0, 1\}$*. Sample a 2-out-of-2 sharing of* σ *and* r*, denoted* $(\sigma_0, \sigma_1) \leftarrow \mathsf{Share}_b(\sigma)$ *and* $(r_0, r_1) \leftarrow \mathsf{Share}_1(r)$*. Sample two sources* $w_0 \leftarrow \{0,1\}^a$ *and* $w_1 \leftarrow \{0,1\}^a$*, and compute* $v_i := \mathsf{acExt}(w_i; \sigma)$*. For* $i \in \{0, 1\}$*, the* i*-th share is defined to be* $\mathbf{s}_i := (w_i, s'_i \oplus v_i \oplus r, \sigma_i, r_i)$.

- (*$\mathsf{LRRecover}$*) *To recover a secret given the shares* $\mathbf{s}_0 = (w_0, e_0, \sigma_0, r_0)$ *and* $\mathbf{s}_1 = (w_1, e_1, \sigma_1, r_1)$*, we first recover* $\sigma = \mathsf{Recover}_b(\sigma_0, \sigma_1)$ *and* $r = \mathsf{Recover}_1(r_0, r_1)$*, compute* $s'_i = e_i \oplus \mathsf{acExt}(w_i; \sigma) \oplus r$ *for* $i \in \{0, 1\}$*, and output* $\mathsf{Recover}_1(s'_0, s'_1)$.

The following theorem is a special case of [SV19, Theorem 3.3].

Theorem 4.7. *Construction 4.6 is a 2-party* $(\mu, 4\epsilon)$*-local leakage-resilient secret-sharing scheme.*

Instantiation. Looking forward, we use the secret sharing scheme to construct leakage-resilient MPC in a $\mathcal{G}$-hybrid model for some functionality $\mathcal{G}$ that takes as an input shares of the leakage-resilient secret-sharing scheme. Our final construction will have exponential dependency on the share size, and therefore, it will be useful to focus on the case where the share-length is *logarithmic* in the (statistical) security parameter. We continue with a concrete instantiation of the secret sharing scheme.

By Corollary A.4 there is an explicit average-case $(a - \mu, \epsilon/4)$-strong extractor $\mathsf{acExt} : \{0,1\}^a \times \{0,1\}^{2a} \to \{0,1\}$ for every ϵ that satisfies $\log(4/\epsilon) \leq (a - \mu + 1)/2$. For a security parameter κ, leakage $\mu = \lfloor \log(\kappa) \rfloor$ and error $\epsilon = 1/\kappa$, we can set $a = \lceil 3\log(\kappa) + 3 \rceil$ to obtain the following theorem.

Theorem 4.8. *There is a polynomial-time generator* Gen *that on input* κ *(represented in binary) outputs the description of a* $(\mu = \lfloor \log(\kappa) \rfloor, \epsilon = 1/\kappa)$*-local leakage resilient secret sharing scheme* $(\mathsf{LRShare}, \mathsf{LRRecover})$*. The sharing and recovering algorithms are represented by circuits of size* $\mathrm{polylog}(\kappa)$ *and the share-length is* $\ell_{\mathsf{LRSS}} = \lceil 9\log(\kappa) + 15 \rceil$.

Special properties. We observe that our secret sharing scheme satisfies the following special properties.

- There is a randomized algorithm LRRand samples uniformly random strings, such that

$$(\mathsf{LRRand}_A, \mathsf{LRRand}_B) \equiv \mathsf{LRShare}(b; \mathbf{r}),$$

  where the LRRand_A and LRRand_B are two independent invocations of LRRand, b is a random bit, and $\mathbf{r}$ is the internal randomness of $\mathsf{LRShare}$. In particular, every pair of string $\mathbf{s}_0, \mathbf{s}_1 \in \{0,1\}^{\ell_{\mathsf{LRSS}}}$ is in the support of $\mathsf{LRShare}(b)$ for some bit $b \in \{0,1\}$.

- There is a randomized algorithm $\mathsf{LRPartial}$ that samples a uniformly random string, and a deterministic algorithm $\mathsf{LRUpdate}$ such that for every $s \in \{0,1\}$,

$$(\mathbf{z}_A, (\mathbf{z}'_B, \mathsf{LRUpdate}(s, \mathbf{z}_A, \mathbf{z}'_B)) \equiv \mathsf{LRShare}(s; \mathbf{r}),$$

  where $\mathbf{z}_A \leftarrow \mathsf{LRRand}$, $\mathbf{z}'_B \leftarrow \mathsf{LRPartial}$. We call $\mathbf{z}'_B$ a *partial share*. Similarly, it holds that

$$(\mathbf{z}_B, (\mathbf{z}'_A, \mathsf{LRUpdate}(s, \mathbf{z}_B, \mathbf{z}'_A)) \equiv \mathsf{LRShare}(s; \mathbf{r}),$$

  where $\mathbf{z}_B \leftarrow \mathsf{LRRand}$, $\mathbf{z}'_A \leftarrow \mathsf{LRPartial}$.

Indeed, for the first property we note that if we sample the two shares $(\mathbf{s}_0, \mathbf{s}_1)$ uniformly at random, then they have the same distribution as $\mathsf{LRShare}(b; \mathbf{r})$ where b is a uniformly distributed bit. For the second property, we let $\mathsf{LRPartial}$ sample (w_1, σ_1, r_1) uniformly at random, and $\mathsf{LRUpdate}(s, (w_0, e_0, \sigma_0, r_0), (w_1, \sigma_1, r_1))$ computes $r = r_0 \oplus r_1$, $\sigma = \sigma_0 \oplus \sigma_1$, $s'_0 = e_0 \oplus \mathsf{acExt}(w_0; \sigma) \oplus r$, $s'_1 = s \oplus s'_0$ and $v_1 = \mathsf{acExt}(w_1; \sigma)$, and output $e_1 = s'_1 \oplus v_1 \oplus r$. It is not hard to verify that when $(w_0, e_0, \sigma_0, r_0)$ and (w_1, σ_1, r_1) are uniformly distributed, then $((w_0, e_0, \sigma_0, r_0), (w_1, e_1, \sigma_1, r_1)) \equiv \mathsf{LRShare}(s)$.

We emphasize that $\mathsf{LRRand}_A, \mathsf{LRRand}_B$ and $\mathsf{LRPartial}$ are "public coin" algorithms, i.e., they simply output a uniformly random string, and have no internal randomness that has to be kept private. This is crucial when we use them as part of a leakage-resilient protocol, since in general the leakage functions might leak information about the internal randomness of the algorithms. Finally, we may assume that the generator Gen promised in Theorem 4.8 also outputs the description of $(\mathsf{LRRand}, \mathsf{LRPartial}, \mathsf{LRUpdate})$.

Additional notation. Given a vector $\mathbf{s} \in \{0,1\}^n$ we denote by $\mathsf{LRvShare}(\mathbf{s})$ the procedure that shares $\mathbf{s}$ bit-by-bit, i.e., it first samples $(\mathbf{s}_{i,0}, \mathbf{s}_{i,1}) \leftarrow \mathsf{LRShare}(s_i)$ and outputs $((\mathbf{s}_{i,0})_{i\in[n]}, (\mathbf{s}_{i,1})_{i\in[n]})$. We extend this convention naturally to $\mathsf{LRvRecover}$, $\mathsf{LRvPartial}$ and $\mathsf{LRvUpdate}$.

We also mention that in some protocols we will use two distinct secret sharing schemes, denoted Σ_1 and Σ_2. To distinguish between the procedures of Σ_1 and the procedures of Σ_2 we will add them as a subscript to the procedure, e.g., we will denote by $\mathsf{LRShare}_{\Sigma_1}$ the sharing procedure of Σ_1.

4.3 Leakage-Resilient Composable Protocols

At a high level, when we compose leakage-resilient protocols, we use the outputs of a sub-protocol A as the inputs of some sub-protocol B. Syntactically, this means that our sub-protocols

will always deliver to the parties a (leakage-resilient) secret-sharing of the intermediate outputs. Maintaining security under composition here, requires some care. Naively, in the simulation we would use the simulator of A to simulate the view from A's execution, and then the simulator of B to simulate the view from B's execution. However, there is a mismatch: the simulator of B also re-samples the encoding of B's inputs, so they will not necessarily be equal to the outputs from A. To solve this problem we require that the simulator of B will be *independent of the inputs*. Of course, this means that the simulator cannot simulate B's inputs, but this is good enough, as the inputs would be taken to be A's outputs. In fact, in some cases we would require that the simulator is also independent of the outputs, that will be sampled separately.

Our n-party protocol will be obtained by composing two-party protocols and a special form of four-party protocols, called senders-receivers protocols. We emphasize that, since our composable protocols will be used as sub-protocols in our construction, they do not include input encoding, and we also only consider the case where they do not require randomness corelation (but rather, they use two-party solitary-output functionalities). We continue with some basic definitions, and then explain the definition of two-party composable protocols (Section 4.3.1) and of senders-receivers composable protocols (Section 4.3.2).

Non-standard terminology. For an n-party protocol π, and vectors of parties' inputs $\mathbf{x} = (x_1, \ldots, x_n)$ and parties randomness $\mathbf{r} = (r_1, \ldots, r_n)$, we define the *partial view* of a party p, denoted $\mathsf{pView}_p(\mathbf{x}; \mathbf{r})$, to be the values of all the variables that belong to the party, except for the inputs and outputs that are given to p in an execution of π over the $\mathbf{x}$ and $\mathbf{r}$. We denote the output of p by $\mathsf{out}_p(\mathbf{x}; \mathbf{r})$. When $\mathbf{r}$ is sampled uniformly we treat $\mathsf{pView}_p(\mathbf{x})$ and $\mathsf{out}_p(\mathbf{x})$ as random variables and when $\mathbf{x}$ is clear from the context, we omit it from the notation.

4.3.1 On Two-Party Composable Protocols

We continue with the definition of two-party composable protocols. In the definition we consider a *randomized* circuit, i.e., a Boolean circuit equipped with *randomness gates* that take no inputs and output a random bit. As already mentioned the protocol will deliver to the parties a *secret sharing* of the output.

Definition 4.9. *Let $\Sigma = (\mathsf{Share}, \mathsf{Recover})$ be a two-party secret-sharing scheme. A two-party protocol π computes a randomized circuit $f : \{0,1\}^{\ell_A} \times \{0,1\}^{\ell_B} \to \{0,1\}^m$ with* (μ, ϵ)-composable leakage-resilient passive security *with respect to Σ, if there exist a simulator such that the following holds.*

- (Perfect correctness) *For every inputs $a \in \{0,1\}^{\ell_A}$ to Alice and $b \in \{0,1\}^{\ell_B}$ to Bob, in an execution of π the parties output vectors $(\mathbf{y}_A^1, \ldots, \mathbf{y}_A^m)$ and $(\mathbf{y}_B^1, \ldots, \mathbf{y}_B^m)$ that form a Σ-sharing of $(y^1, \ldots, y^m)$, i.e., $y^i = \mathsf{Recover}(\mathbf{y}_A^i, \mathbf{y}_B^i), \forall i$, and $(y^1, \ldots, y^m) \equiv f(a, b)$.*
- (Statistical privacy) *For every pair of inputs $a \in \{0,1\}^{\ell_A}$ and $b \in \{0,1\}^{\ell_B}$, every leakage functions τ_A, τ_B that output μ bits, every string $\mathbf{y} \in \{0,1\}^m$, and every set $Z \subsetneq \{A, B\}$ it holds that*

$$(\mathsf{pView}_p, \mathsf{out}_p)_{p \in Z}, (\tau_p(\mathsf{pView}_p, \mathsf{out}_p))_{p \notin Z} \equiv_\epsilon (\mathsf{pView}'_p, \mathsf{out}'_p)_{p \in Z}, (\tau_p(\mathsf{pView}'_p, \mathsf{out}'_p))_{p \notin Z} \quad (1)$$

where on the LHS the partial views $\mathsf{pView}_A, \mathsf{pView}_B$ and the outputs $\mathsf{out}_A, \mathsf{out}_B$ were jointly sampled by executing π with inputs (a, b) and uniform randomness for A and B, and on the RHS the outputs are sampled by $(\mathsf{out}'_A, \mathsf{out}'_B) \leftarrow \mathsf{vShare}_\Sigma(\mathbf{y})$, and the partial views $\mathsf{pView}'_A, \mathsf{pView}'_B$ were generated by $\mathsf{Sim}(Z, (\mathsf{out}'_A, \mathsf{out}'_B))$.

The protocol computes f with strong *(μ,ϵ)-composable leakage-resilient passive security if in addition it satisfies the following conditions:*

- (Real-world output independence) *For every pair of inputs $a \in \{0,1\}^{\ell_A}$ and $b \in \{0,1\}^{\ell_B}$, every leakage functions τ_A, τ_B that output μ bits, every set $Z \subsetneq \{A,B\}$ and every fixed outputs $(\mathsf{out}_A, \mathsf{out}_B)$ in the support of $\mathsf{vShare}_\Sigma(f(a,b))$, it holds that*

$$(\mathsf{pView}_p)_{p\in Z}, (\tau_p(\mathsf{pView}_p))_{p\notin Z} \equiv_\epsilon (\mathsf{pView}'_p)_{p\in Z}, (\tau_p(\mathsf{pView}'_p))_{p\notin Z}$$

  where on the LHS $(\mathsf{pView}_A, \mathsf{pView}_B)$ are jointly sampled by executing π with inputs (a,b) and uniform randomness for A and B, and on the RHS $(\mathsf{pView}_A, \mathsf{pView}_B)$ are sampled from an execution of π with inputs (a,b) conditioned on the outputs being $(\mathsf{out}_A, \mathsf{out}_B)$.

- (Simulation output independence) *The simulator* Sim *takes only the set Z as an input, and does not require the outputs generated by the output encoder.*

Let π be a two-party protocol computing a circuit f, let $Z \subseteq \{A,B\}$, let (a,b) be possible inputs to Alice and Bob, and let $\vec{\tau} = (\tau_1, \ldots, \tau_n)$ be leakage functions. We define the *partial real-world experiment*, denoted $\mathsf{pReal}_{\pi,Z,\vec{\tau}}(a,b)$ to be the random variables on the LHS of Equation 1. Similarly, for a string $\mathbf{y} \in \{0,1\}^m$, we define the *partial ideal-world experiment*, denoted $\mathsf{pIdeal}_{\pi,Z,\vec{\tau}}(\mathbf{y})$, to be the random variables on the RHS of Equation 1.

4.3.2 On Senders-Receivers Composable Protocols

A four-party *senders-receivers protocol* is a protocol π in which there are two *senders* S_0 and S_1 and two *receivers* R_0 and R_1, so that only the receivers have outputs (while both senders and receivers have inputs). We extend this notion to the case where there are only two or three parties by letting each party take more than one role as long as no party plays both S_0 and S_1 and no party plays both R_0 and R_1. That is, a sender may also be a receiver (e.g., when S_0 and R_1 is the same party), and it is even possible that the two senders are also the two receivers (e.g., S_0 is R_0 and S_1 is R_1), but we always assume that we have two distinct senders and two distinct receivers.

Formally, we denote the set of parties by $\mathcal{P} = \{P_1, \ldots, P_n\}$ for $2 \le n \le 4$, and let $\phi : \{S_0, S_1, R_0, R_1\} \to \mathcal{P}$ be a sujective mapping with $\phi(S_0) \neq \phi(S_1)$ and $\phi(R_0) \neq \phi(R_1)$, that indicates which party takes the role of S_0, S_1, R_0 and R_1. The protocol is specified by the tuple $(\pi, \mathcal{P}, \phi)$ where π is a 4-party protocol over (S_0, S_1, R_0, R_1). Note that once $\mathcal{P}$ and ϕ are given, statements about the information that π provides to the "roles" S_0, S_1, R_0, R_1 translate into statements about the information that actual parties receive. In particular, the outputs of $P_1, \ldots, P_n$ are fully determined by indicating the outputs of the two receivers. In this context, we will only be interested in a *deterministic* functionalities f.

Definition 4.10. *Let $2 \le n \le 4$ be the number of parties, let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be the parties, and let $\phi : \{S_0, S_1, R_0, R_1\} \to \mathcal{P}$ be the surjective role-assigning mapping. Let Σ be a two-party secret-sharing scheme. An n-party senders-receivers protocol $(\pi, \mathcal{P}, \phi)$ computes a deterministic circuit $f : (\{0,1\}^{\ell_{S_0}} \times \{0,1\}^{\ell_{S_1}}) \times (\{0,1\}^{\ell_{R_0}} \times \{0,1\}^{\ell_{R_1}}) \to \{0,1\}^m$ with* (μ,ϵ)-composable leakage-resilient passive security *with respect to Σ, if there exist a simulator* Sim *such that the following holds.*

- (Perfect correctness) *For every inputs $a_0 \in \{0,1\}^{\ell_{S_0}}$ and $a_1 \in \{0,1\}^{\ell_{S_1}}$ to the senders, and $b_0 \in \{0,1\}^{\ell_{R_0}}$ and $b_1 \in \{0,1\}^{\ell_{R_1}}$ to the receivers, in an execution of π the receivers output vectors $(\mathbf{y}_0^1, \ldots, \mathbf{y}_0^m)$ and $(\mathbf{y}_1^1, \ldots, \mathbf{y}_1^m)$ such that $(\mathbf{y}_0^i, \mathbf{y}_1^i)$ is a Σ-sharing of a bit y^i, and $(y^1, \ldots, y^m) = f((a_0,a_1),(b_0,b_1))$.*

- (Statistical privacy) *For every inputs* $a_0 \in \{0,1\}^{\ell_{S_0}}$ *and* $a_1 \in \{0,1\}^{\ell_{S_1}}$ *to the senders, and* $b_0 \in \{0,1\}^{\ell_{R_0}}$ *and* $b_1 \in \{0,1\}^{\ell_{R_1}}$ *to the receivers, every leakage functions* $\tau_{P_1}, \ldots, \tau_{P_n}$ *that output* μ *bits, every* $\mathbf{y} \in \{0,1\}^m$ *and every set* $Z \subsetneq \mathcal{P}$ *it holds that*

$$\begin{aligned}&(\mathsf{pView}_{P_i}, \mathsf{out}_{P_i})_{P_i \in Z}, (\tau_{P_i}(\mathsf{pView}_{P_i}, \mathsf{out}_{P_i}))_{P_i \in \mathcal{P} \setminus Z} \\ &\equiv_\epsilon (\mathsf{pView}'_{P_i}, \mathsf{out}'_{P_i})_{P_i \in Z}, (\tau_{P_i}(\mathsf{pView}'_{P_i}, \mathsf{out}'_{P_i}))_{P_i \in \mathcal{P} \setminus Z}\end{aligned} \tag{2}$$

where on the LHS the partial views $\mathsf{pView}_{P_1}, \ldots, \mathsf{pView}_{P_n}$ *and the outputs* $\mathsf{out}_{P_1}, \ldots, \mathsf{out}_{P_n}$ *are jointly sampled by executing* π *with input* a_0 *to* S_0*,* a_1 *to* S_1*,* b_0 *to* R_0 *and* b_1 *to* R_1 *and uniformly chosen randomness for each party; and on the RHS the random variables are sampled as follows*

1. *if* $\phi(R_0), \phi(R_1) \in Z$ *then set* $\mathsf{out}' := f((a_0, a_1), (b_0, b_1))$*, and otherwise set* $\mathsf{out}' := \mathbf{y}$*,*
2. *sample* $(\mathsf{out}'_0, \mathsf{out}'_1) \leftarrow \mathsf{vShare}_\Sigma(\mathsf{out}')$*,*
3. *the partial views* $(\mathsf{pView}'_{P_1}, \ldots, \mathsf{pView}'_{P_n})$ *are sampled by* $\mathsf{Sim}(Z, \mathsf{out}', \mathsf{aux})$*, where* aux *contains* (a_0, a_1) *if* $\phi(S_0), \phi(S_1) \in Z$*, and* (b_0, b_1) *if* $\phi(R_0), \phi(R_1) \in Z$*.*

- (Real-world output independence) *For every inputs* $a_0 \in \{0,1\}^{\ell_{S_0}}$ *and* $a_1 \in \{0,1\}^{\ell_{S_1}}$ *to the senders, and* $b_0 \in \{0,1\}^{\ell_{R_0}}$ *and* $b_1 \in \{0,1\}^{\ell_{R_1}}$*, it holds that*

$$(\mathsf{pView}_p, \mathsf{out}_p)_{p \in \mathcal{P}} \equiv (\mathsf{pView}'_p, \mathsf{out}'_p)_{p \in \mathcal{P}}$$

where on the LHS $(\mathsf{pView}_p, \mathsf{out}_p)_{p \in \mathcal{P}}$ *are sampled in an honest execution of* π *with the inputs we've fixed; and on the RHS* $(\mathsf{pView}'_p)_{p \in \mathcal{P}}$ *are sampled in an honest execution of* π *with the inputs we've fixed, while the outputs* $(\mathsf{out}'_p)_{p \in \mathcal{P}}$ *are sampled in the following way: first we sample* $(\mathbf{s}_0, \mathbf{s}_1) \leftarrow \mathsf{vShare}_\Sigma(f((a_0, a_1), (b_0, b_1)))$ *and then we set* $\mathsf{out}'_{\phi(R_0)} := \mathbf{s}_0$*,* $\mathsf{out}'_{\phi(R_1)} := \mathbf{s}_1$*, and the remaining outputs are set to be* $\perp$*.*

Let $(\pi, \mathcal{P}, \phi)$ be a senders-receivers protocol computing a circuit f, let $Z \subseteq \mathcal{P}$, let $((a_0, a_1), (b_0, b_1))$ be the inputs of the senders and the receivers, and let $\vec{\tau} = (\tau_1, \ldots, \tau_n)$ be leakage functions. We define the *partial real-world experiment*, denoted $\mathsf{pReal}_{\pi, Z, \vec{\tau}}((a_0, a_1), (b_0, b_1))$ to be the random variables on the LHS of Equation 2. Similarly, for a string $\mathbf{y}$, we define the *partial ideal-world experiment*, denoted $\mathsf{pIdeal}_{\pi, Z, \vec{\tau}}((a_0, a_1), (b_0, b_1), \mathbf{y})$, to be the random variables on the RHS of Equation 2. (Observe that the simulated views might depend on the real output $f((a_0, a_1), (b_0, b_1))$ if both receivers are in Z.)

4.3.3 Strongly-Composable Two-Party Protocols

We continue with a description of a strongly-composable two-party protocol, that will be an important building block in the final construction. We describe the protocol with respect to a general two-party secret sharing scheme Σ that satisfies the special properties of our construction (see Section 4.2). At a high level, our protocol follows the two-party GMW protocol [GMW87, GMW91], except that the additive 2-out-of-2 secret sharing is replaced with leakage resilient secret sharing, and given shares of the input of some gate in the computation, the shares of the outputs are computed by a call to a two-party solitary-output functionality. We continue with the definition of the two-party functionalities.

Functionality $\mathcal{F}^*_\Sigma$

The functionality is parameterized by an operation $* \in \{+, \times\}$ over $\mathbb{F}_2$, and a two-party secret-sharing scheme Σ.

- **Inputs:** The functionality takes from Alice Σ-shares $(\mathbf{a}_A, \mathbf{b}_A, \mathbf{c}_A)$ and from Bob $(\mathbf{a}_B, \mathbf{b}_B, \mathbf{c}'_B)$, where $\mathbf{c}'_B$ is a partial share.
- **Computation:** The functionality computes $a = \mathsf{LRRecover}(\mathbf{a}_A, \mathbf{a}_B)$, $b = \mathsf{LRRecover}(\mathbf{b}_A, \mathbf{b}_B)$, and computes $c = a * b$. Then, the functionality computes $\mathbf{c}_B := (\mathbf{c}'_B, \mathsf{LRUpdate}(c, \mathbf{c}_A, \mathbf{c}'_B))$.
- **Outputs:** The functionality returns $\mathbf{c}_B$ to Bob.

Figure 2: Functionality $\mathcal{F}^*_\Sigma$

Functionality $\mathcal{F}^{\mathsf{Share}}_{\Sigma,P}$

The functionality is parameterized by a party P which is either Alice or Bob, and a two-party secret-sharing scheme Σ.

- **Inputs:** The functionality takes from Alice a Σ-share $\mathbf{s}_A$ and from Bob a partial share $\mathbf{s}'_B$. In addition, the functionality takes from P a bit s.
- **Computation:** The functionality computes $\mathbf{s}_B := (\mathbf{s}'_B, \mathsf{LRUpdate}(s, \mathbf{s}_A, \mathbf{s}'_B))$.
- **Outputs:** The functionality returns $\mathbf{s}_B$ to Bob.

Figure 3: Functionality $\mathcal{F}^{\mathsf{Share}}_{\Sigma,P}$

We continue with the description of the protocol.

Protocol $\mathsf{LR2PC}_\Sigma(f)$

The protocol is parameterized by a two-party secret sharing scheme Σ and a randomized circuit $f : \{0,1\}^{\ell_A} \times \{0,1\}^{\ell_B} \to \{0,1\}^m$. The protocol involves two parties: Alice and Bob.

- **Primitives:** The two-party solitary-output functionalities $\mathcal{F}^+_\Sigma$, $\mathcal{F}^\times_\Sigma$, $\mathcal{F}^{\mathsf{Share}}_{\Sigma,A}$ and $\mathcal{F}^{\mathsf{Share}}_{\Sigma,B}$ from Figure 2 and Figure 3.
- **Inputs:** Alice holds an input $\mathbf{a} \in \{0,1\}^{\ell_A}$ and Bob holds an input $\mathbf{b} \in \{0,1\}^{\ell_B}$.
- **Inputs sharing:** For every input wire, Alice samples a share $\mathbf{x}_A \leftarrow \mathsf{LRRand}$ and Bob samples a partial share $\mathbf{x}'_B \leftarrow \mathsf{LRPartial}$. If the input wire belongs to Alice and corresponds to the input bit a_j, then Alice and Bob call $\mathcal{F}^{\mathsf{Share}}_{\Sigma,A}$ with inputs $(\mathbf{x}_A, a_j)$ for Alice and $\mathbf{x}'_B$ for Bob. Otherwise, if the input wire belongs to Bob and corresponds to the input bit b_j, then Alice and Bob call $\mathcal{F}^{\mathsf{Share}}_{\Sigma,B}$ with inputs $\mathbf{x}^i_A$ for Alice and $(\mathbf{x}'_B, b_j)$ for Bob. If the wire is a constant wire, with value $b \in \{0,1\}$, then Alice and Bob call $\mathcal{F}^{\mathsf{Share}}_{\Sigma,A}$ with inputs $(\mathbf{x}_A, b)$ for Alice and $\mathbf{x}'_B$ for Bob. Bob takes the output of the functionality, denoted $\mathbf{x}_B$, to be its share of the input wire.
- **Randomness Sharing:** For every wire outgoing from a randomness gate, Alice samples a share $\mathbf{x}_A \leftarrow \mathsf{LRRand}$ and Bob samples a share $\mathbf{x}_B \leftarrow \mathsf{LRRand}$.

- **Computation:** Alice and Bob compute the circuit f gate-by-gate in topological ordering. At gate g, Alice and Bob hold a Σ-sharing of the input wires a and b, denoted $(\mathbf{a}_A, \mathbf{a}_B)$ and $(\mathbf{b}_A, \mathbf{b}_B)$, and they compute a Σ-sharing of the output wire c, denoted $(\mathbf{c}_A, \mathbf{c}_B)$. They do so in the following way.

  Assume that the gate g computes the operation $* \in \{+, \times\}$. Alice samples $\mathbf{c}_A \leftarrow \mathsf{LRRand}$ and Bob samples $\mathbf{c}'_A \leftarrow \mathsf{LRPartial}$. Alice and Bob call $\mathcal{F}^*_\Sigma$ with inputs $(\mathbf{a}_A, \mathbf{b}_A, \mathbf{c}_A)$ for Alice, and inputs $(\mathbf{a}_B, \mathbf{b}_B, \mathbf{c}'_B)$ for Bob, and Bob receives $\mathbf{c}_B$ as an output from $\mathcal{F}^*_\Sigma$.
- **Outputs:** Let $w_1, \ldots, w_m$ be the output wires, and let $(\mathbf{w}^1_A, \ldots, \mathbf{w}^m_A)$ and $(\mathbf{w}^1_B, \ldots, \mathbf{w}^m_B)$ be the corresponding shares of Alice and Bob. Alice outputs $(\mathbf{w}^1_A, \ldots, \mathbf{w}^m_A)$ and Bob outputs $(\mathbf{w}^1_B, \ldots, \mathbf{w}^m_B)$.

Figure 4: Protocol $\mathsf{LR2PC}_\Sigma(f)$

Theorem 4.11. *Let Σ be a two-party (μ, ϵ)-local leakage-resilient secret sharing scheme, and let $f : \{0,1\}^{\ell_A} \times \{0,1\}^{\ell_B} \to \{0,1\}^m$ be a randomized circuit with W wires. Then Protocol $\mathsf{LR2PC}_\Sigma(f)$ computes f with strong $(\mu, W \cdot \epsilon)$-composable leakage-resilient passive security in the $(\mathcal{F}^+_\Sigma, \mathcal{F}^\times_\Sigma, \mathcal{F}^{\mathsf{Share}}_{\Sigma,A}, \mathcal{F}^{\mathsf{Share}}_{\Sigma,B})$-hybrid model.*

We prove Theorem 4.11 in Section B.1.

5 Leakage-Resilient MPC: Basic Construction

In this section we present our basic construction of leakage-resilient MPC. In our construction, we do not use correlated randomness, but rather assume that the parties have access to "simple" two-party solitary-output functionalities. This presentation will be useful when we transform our leakage-resilient MPC into leakage-resilient NPSS in Section 7. We mention that in Section 6 we explain how to transform our protocol into a protocol with pairwise correlated randomness and no calls to ideal functionalities.

As discussed in Section 2, our protocol is obtained by transforming an MPC protocol π' in the OLE-hybrid model[6] with standard (non-leakage-resilient) security into a protocol π with leakage-resilient security, by emulating every party in π' by some (appropriately chosen) pair of parties in π. In more details, for every party w_i in π' there is a pair (p_{i_1}, p_{i_2}) in π that holds a leakage-resilient secret sharing of w_i's view, and updates the secret-sharing view of w_i by following the instructions of π'. Therefore, in Section 5.1 we first explain how, given a secret sharing of w_i's view and some statement of π' that corresponds to w_i, the pair (p_{i_1}, p_{i_2}) can update the secret-shared view in a leakage-resilient way. Our final protocol is presented in Section 5.2. Throughout the section, our results are presented with respect to a two-party secret sharing scheme Σ that satisfies the special properties presented in Section 4.2.

5.1 Compiling Protocol Instructions

In this section we explain how every statement in the original protocol π' is compiled in π. Most statements, that involve only a single party, are translated into relatively simple two-party leakage-resilient protocols that will be analyzed as part of our main construction in Section 5.2.

[6]The use of the OLE-hybrid model instead of the OT-hybrid model is just for convenience, since the two models are equivalent (an OT call can be emulated by an OLE call and vice versa.)

Statements that involve two parties, such as OLE statements and transmit statement, will be compiled into more complicated senders-receivers protocol. We note that it is enough to focus on OLE statements, as any transmit statement can be reduced to an OLE statement.

5.1.1 Compiling Local Operations

We begin with the compilation of a local operation that is performed by the parties, i.e., addition or multiplication over $\mathbb{F}_2$. Let $* \in \{+, \times\}$ be addition or multiplication over $\mathbb{F}_2$. Let $f_\Sigma^{\mathsf{LROp}(*)}$ be the circuit that takes as an input two pairs of Σ-shares, $(\mathbf{x}_0, \mathbf{y}_0)$ from Alice and $(\mathbf{x}_1, \mathbf{y}_1)$ from Bob, computes $x = \mathsf{LRRecover}_\Sigma(\mathbf{x}_0, \mathbf{x}_1)$, and $y = \mathsf{LRRecover}_\Sigma(\mathbf{y}_0, \mathbf{y}_1)$, and outputs $x * y$. Our goal is to construct a protocol that computes $f_\Sigma^{\mathsf{LROp}(*)}$ with composable leakage resilient security.

Protocol $\mathsf{LROp}_\Sigma(*)$

The protocol is parameterized by a two-party secret sharing scheme Σ and an operation $* \in \{+, \times\}$. The protocol involves two parties: Alice and Bob.

- **Primitives:** The two-party solitary-output functionality $\mathcal{F}_\Sigma^*$ from Figure 2.
- **Inputs:** Alice and Bob hold Σ-shares $(\mathbf{x}_0, \mathbf{y}_0)$ and $(\mathbf{x}_1, \mathbf{y}_1)$, respectively, of two bits $x, y \in \{0, 1\}$.
- **Computation:** The parties do as follows.
  1. Alice samples random Σ-share $\mathbf{z}_0 \leftarrow \mathsf{LRRand}$. Bob samples a random partial share $\mathbf{z}_1' \leftarrow \mathsf{LRPartial}$.
  2. The parties call $\mathcal{F}_\Sigma^*$, where Alice inputs $(\mathbf{x}_0, \mathbf{y}_0)$ and $\mathbf{z}_0$, and Bob inputs $(\mathbf{x}_1, \mathbf{y}_1)$ and $\mathbf{z}_1'$. Denote the output of Bob by $\mathbf{z}_1$.
- **Outputs:** Alice outputs $\mathbf{z}_0$ and Bob outputs $\mathbf{z}_1$.

Figure 5: Protocol $\mathsf{LROp}_\Sigma(*)$

The following lemma follows immediately from the local leakage-resilience of Σ.

Lemma 5.1. *Let Σ be a two-party (μ, ϵ)-local leakage-resilient secret sharing scheme, and let $* \in \{+, \times\}$. Then Protocol $\mathsf{LROp}_\Sigma(*)$ computes $f_\Sigma^{\mathsf{LROp}(*)}$ with (μ, ϵ)-composable leakage-resilient passive security in the $\mathcal{F}_\Sigma(*)$-hybrid model.*

5.1.2 Compiling Output Statement

We continue with the compilation of an output statement. For this we need the following two-party ideal functionality, that takes two shares and recovers the shared value.

Functionality $\mathcal{F}_\Sigma^{\mathsf{out}}$

The functionality is parameterized by a two-party secret sharing scheme Σ. The functionality involves two parties: Alice and Bob.

- **Inputs:** Alice and Bob hold Σ-shares $\mathbf{x}_0$ and $\mathbf{x}_1$, respectively, of a bit $x \in \{0,1\}$.
- **Computation:** The functionality recovers $x = \mathsf{LRRecover}_\Sigma(\mathbf{x}_0, \mathbf{x}_1)$.
- **Outputs:** The functionality returns x to Bob.

Figure 6: Functionality $\mathcal{F}_\Sigma^{\mathsf{out}}$

In the protocol, we simply use $\mathcal{F}_\Sigma^{\mathsf{out}}$ to recover the output. We continue with a formal definition of the protocol.

Protocol LROut_Σ

The protocol is parameterized by a two-party secret sharing scheme Σ. The protocol involves two parties: Alice and Bob.

- **Primitives:** The two-party solitary-output functionality $\mathcal{F}_\Sigma^{\mathsf{out}}$ from Figure 6.
- **Inputs:** Alice and Bob hold Σ-shares $\mathbf{x}_0$ and $\mathbf{x}_1$, respectively.
- **Computation:** Alice and Bob call $\mathcal{F}_\Sigma^{\mathsf{out}}$ with inputs $\mathbf{x}_0$ and $\mathbf{x}_1$, and Bob receives the output b. Bob sends b to Alice.
- **Outputs:** Alice and Bob output b.

Figure 7: Protocol LROut_Σ

5.1.3 Compiling Abort Statement

The compilation of an abort statement is very similar to the compilation of an output statement. That is, we recover the shared-bit that decides whether the parties should raise the abort flag, and act according to its value.

Protocol $\mathsf{LRAbort}_\Sigma$

The protocol involves two parties, P_0 and P_1.

- **Primitives:** The two-party solitary-output functionality $\mathcal{F}_\Sigma^{\mathsf{out}}$ from Figure 6.
- **Inputs:** Alice and Bob hold Σ-shares $\mathbf{x}_0$ and $\mathbf{x}_1$, respectively.
- **Computation:** Alice and Bob call $\mathcal{F}_\Sigma^{\mathsf{out}}$ with inputs $\mathbf{x}_0$ and $\mathbf{x}_1$, and Bob receives the output b. If $b = 1$ then Bob raises the abort flag.

Figure 8: Protocol $\mathsf{LRAbort}_\Sigma$

5.1.4 Compiling OLE Statements

Finally, we explain how to compile OLE statements. Recall that the OLE functionality takes from Alice two bits (a, b) and from Bob a single bit x and returns the bit $a \cdot x \oplus b$ to Bob. We first need

to define the following two-party solitary-output ideal functionalities.

Functionality $\mathcal{F}_{\mathbf{z}}^{\mathsf{condTrans}}$

The functionality is parameterized by a binary string $\mathbf{z} \in \{0,1\}^{\ell}$. The functionality involves two parties: a sender S and a receiver R.

- **Inputs:** The sender inputs a bit $s \in \{0,1\}$ and the receiver inputs a string $\mathbf{z}' \in \{0,1\}^{\ell}$.
- **Outputs:** If $\mathbf{z}' = \mathbf{z}$ then the functionality outputs b to the receiver. Otherwise, the functionality outputs 0 to the receiver.

Figure 9: Functionality $\mathcal{F}_{\mathbf{z}}^{\mathsf{condTrans}}$

Functionality $\mathcal{F}_{\Sigma_1,\Sigma_2}^{\mathsf{translate}}$

The functionality is parameterized by two two-party secret sharing schemes Σ_1 and Σ_2. The functionality involves two parties: Alice and Bob.

- **Inputs:** Alice inputs a Σ_2-LRSS share $\mathbf{w}_0$, together with a Σ_1-share $\mathbf{m}_0$. Bob inputs a Σ_2-LRSS share $\mathbf{w}_1$, together with a Σ_1-partial share $\mathbf{m}_1'$.
- **Computation:** The functionality recovers $w = \mathsf{LRRecover}_{\Sigma_2}(\mathbf{w}_0, \mathbf{w}_1)$, and computes $\mathbf{m}_1 = (\mathbf{m}_1', \mathsf{LRUpdate}_{\Sigma_1}(w, \mathbf{m}_0, \mathbf{m}_1'))$.
- **Outputs:** The functionality returns $\mathbf{m}_1$ to Bob.

Figure 10: Functionality $\mathcal{F}_{\Sigma_1,\Sigma_2}^{\mathsf{translate}}$

We continue with a formal description of the protocol. Throughout the protocol, we use the following convention: whenever a party calls a two-party functionality both as Alice and Bob, then the party simply computes the output by herself (this happens when the same party takes the role of a sender and a receiver).

We will extensively use the sub-protocol LR2PC defined in Figure 4. Recall that for a two-party secret sharing scheme Σ and a randomized circuit $f : \{0,1\}^{\ell_A} \times \{0,1\}^{\ell_B} \to \{0,1\}^m$, the protocol $\mathsf{LR2PC}_{\Sigma}(f)$ takes an input $\mathbf{a} \in \{0,1\}^{\ell_A}$ from Alice and $\mathbf{b} \in \{0,1\}^{\ell_B}$ from Bob and delivers to the parties a secret-sharing of $f(\mathbf{a}, \mathbf{b})$ under Σ.

Protocol $\mathsf{LROLE}_{\Sigma_1,\Sigma_2}$

The protocol is parameterized by two two-party secret sharing schemes: $\Sigma_1 = (\mathsf{LRShare}_{\Sigma_1}, \mathsf{LRRecover}_{\Sigma_1})$ and $\Sigma_2 = (\mathsf{LRShare}_{\Sigma_2}, \mathsf{LRRecover}_{\Sigma_2})$. We denote the length of the share in Σ_i by ℓ_i. There are two distinct senders S_0, S_1 and two distinct receivers R_0, R_1.

- **Primitives:** The two-party solitary-output ideal functionalities $\mathcal{F}_{\mathbf{z}}^{\mathsf{condTrans}}$ and $\mathcal{F}_{\Sigma_1,\Sigma_2}^{\mathsf{translate}}$ from Figure 9 and Figure 10, and the sub-protocols $\mathsf{LR2PC}_{\Sigma_1}(f_{\mathsf{refresh}})$, $\mathsf{LR2PC}_{\Sigma_2}(f_{\mathsf{matrix}})$ and $\mathsf{LR2PC}_{\Sigma_2}(f_{\mathsf{out}})$, where LR2PC was defined in Figure 4, and the underlying circuits are defined as follow.

- Let f_{refresh} be the deterministic circuit that takes a Σ_1-share $\mathbf{x}_0$ from Alice and a Σ_1-share $\mathbf{x}_1$ from Bob, recovers the secret x by computing $x = \mathsf{LRRecover}_{\Sigma_1}(\mathbf{x}_0, \mathbf{x}_1)$, and outputs x.
- Let f_{matrix} be the randomized circuit that takes a Σ_1-shares $(\mathbf{a}_0, \mathbf{b}_0)$ from Alice and $(\mathbf{a}_1, \mathbf{b}_1)$ from Bob and recovers a and b by computing $a = \mathsf{LRRecover}_{\Sigma_1}(\mathbf{a}_0, \mathbf{a}_1)$ and $b = \mathsf{LRRecover}_{\Sigma_1}(\mathbf{b}_0, \mathbf{b}_1)$. For every possible pair of Σ_1-shares $(\mathbf{z}_0, \mathbf{z}_1)$ the function computes $z = \mathsf{LRRecover}_{\Sigma_1}(\mathbf{z}_0, \mathbf{z}_1)$, computes $y^{\mathbf{z}_0,\mathbf{z}_1} = a \cdot z + b$ and samples $(\mathbf{y}_0^{\mathbf{z}_0,\mathbf{z}_1}, \mathbf{y}_1^{\mathbf{z}_0,\mathbf{z}_1}) \leftarrow \mathsf{LRShare}_{\Sigma_1}(y^{\mathbf{z}_0,\mathbf{z}_1})$. The circuit outputs $(\mathbf{y}_0^{\mathbf{z}_0,\mathbf{z}_1}, \mathbf{y}_1^{\mathbf{z}_0,\mathbf{z}_1})_{\mathbf{z}_0,\mathbf{z}_1 \in \{0,1\}^{\ell_1}}$.
- Let f_{out} be the deterministic circuit that takes from Alice a Σ_1-share $\mathbf{x}'_0$ together with a list of Σ_1-shares $(\mathbf{y}_0^{\mathbf{z}})_{\mathbf{z} \in \{0,1\}^{\ell_1}}$, takes from Bob a Σ_1-share $\mathbf{x}'_1$ together with a list of Σ_1-shares $(\mathbf{y}_1^{\mathbf{z}})_{\mathbf{z} \in \{0,1\}^{\ell_1}}$, computes $b = \mathsf{LRRecover}_{\Sigma_1}(\mathbf{y}_0^{\mathbf{x}'_1}, \mathbf{y}_1^{\mathbf{x}'_0})$, and outputs b.

- **Inputs:** The sender S_i holds Σ_1-shares $(\mathbf{a}_i, \mathbf{b}_i)$. The receiver R_i holds a Σ_1-share $\mathbf{x}_i$.
- **Computation:** The parties do as follows.
  1. *(Refreshing shares)* The receivers execute $\mathsf{LR2PC}_{\Sigma_1}(f_{\mathsf{refresh}})$ with inputs $\mathbf{x}_0$ to R_0 and $\mathbf{x}_1$ to R_1. Denote the output of R_0 by $\mathbf{x}'_0$ and the output of R_1 by $\mathbf{x}'_1$.
  2. *(Generating matrix of evaluations)* The senders execute $\mathsf{LR2PC}_{\Sigma_2}(f_{\mathsf{matrix}})$ with inputs $(\mathbf{a}_0, \mathbf{b}_0)$ to S_0 and $(\mathbf{a}_1, \mathbf{b}_1)$ to S_1. Denote the output of S_i by $(\vec{\mathbf{y}}_{0,i}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{1,i}^{\mathbf{z}_0,\mathbf{z}_1})_{\mathbf{z}_0,\mathbf{z}_1 \in \{0,1\}_{\ell_1}}$, where $\vec{\mathbf{y}}_{b,i}^{\mathbf{z}_0,\mathbf{z}_1}[j]$ is the i-th Σ_2-share of the j-th bit of the Σ_1-share $\mathbf{y}_b^{\mathbf{z}_0,\mathbf{z}_1}$.
  3. *(Transmitting $\mathbf{x}'_0$-row to R_0)* For $i \in \{0,1\}$, S_i and R_0 do as follows. For every $\mathbf{z}_0, \mathbf{z}_1 \in \{0,1\}^{\ell_1}$ we denote $\vec{\mathbf{y}}_{0,i}^{\mathbf{z}_0,\mathbf{z}_1} = (b_{0,i,1}^{\mathbf{z}_0,\mathbf{z}_1}, \ldots, b_{0,i,\ell_1 \cdot \ell_2}^{\mathbf{z}_0,\mathbf{z}_1})$, and for every $j \in [\ell_1 \cdot \ell_2]$ we let S_i and R_0 call to $\mathcal{F}_{\mathbf{z}_0}^{\mathsf{condTrans}}$ with input $b_{0,i,j}^{\mathbf{z}_0,\mathbf{z}_1}$ to S_i and $\mathbf{x}'_0$ to R_0.

     At the end of this step R_0 holds $(\vec{\mathbf{y}}_{0,0}^{\mathbf{x}'_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{0,1}^{\mathbf{x}'_0,\mathbf{z}_1})_{\mathbf{z}_1 \in \{0,1\}^{\ell_1}}$ and recovers $(\mathbf{y}_0^{\mathbf{x}'_0,\mathbf{z}_1})_{\mathbf{z}_1 \in \{0,1\}^{\ell_1}}$, where the j-th bit of $\mathbf{y}_0^{\mathbf{x}'_0,\mathbf{z}_1}$ is given by $\mathsf{LRRecover}_{\Sigma_2}(\vec{\mathbf{y}}_{0,0}^{\mathbf{x}'_0,\mathbf{z}_1}[j], \vec{\mathbf{y}}_{0,1}^{\mathbf{x}'_0,\mathbf{z}_1}[j])$.
  4. *(Transmitting $\mathbf{x}'_1$-column to R_1)* For $i \in \{0,1\}$, S_i and R_1 do as follows. For every $\mathbf{z}_0, \mathbf{z}_1 \in \{0,1\}^{\ell_1}$ we denote $\vec{\mathbf{y}}_{1,i}^{\mathbf{z}_0,\mathbf{z}_1} = (b_{1,i,1}^{\mathbf{z}_0,\mathbf{z}_1}, \ldots, b_{1,i,\ell_1 \cdot \ell_2}^{\mathbf{z}_0,\mathbf{z}_1})$, and for every $j \in [\ell_1 \cdot \ell_2]$ we let S_i and R_1 call to $\mathcal{F}_{\mathbf{z}_1}^{\mathsf{condTrans}}$ with input $b_{1,i,j}^{\mathbf{z}_0,\mathbf{z}_1}$ to S_i and $\mathbf{x}'_1$ to R_1.

     At the end of this step R_1 holds $(\vec{\mathbf{y}}_{1,0}^{\mathbf{z}_0,\mathbf{x}'_1}, \vec{\mathbf{y}}_{1,1}^{\mathbf{z}_0,\mathbf{x}'_1})_{\mathbf{z}_0 \in \{0,1\}^{\ell_1}}$ and recovers $(\mathbf{y}_1^{\mathbf{z}_0,\mathbf{x}'_1})_{\mathbf{z}_1 \in \{0,1\}^{\ell_1}}$, where the j-th bit of $\mathbf{y}_1^{\mathbf{z}_0,\mathbf{x}'_1}$ is given by $\mathsf{LRRecover}_{\Sigma_2}(\vec{\mathbf{y}}_{1,0}^{\mathbf{z}_0,\mathbf{x}'_1}[j], \vec{\mathbf{y}}_{1,1}^{\mathbf{z}_0,\mathbf{x}'_1}[j])$.
  5. *(Computing the output)* The receivers execute $\mathsf{LR2PC}_{\Sigma_2}(f_{\mathsf{out}})$ with inputs $\mathbf{x}'_0$ and $(\mathbf{y}_0^{\mathbf{x}'_0,\mathbf{z}_1})_{\mathbf{z}_1 \in \{0,1\}^{\ell_1}}$ to S_0, and $\mathbf{x}'_1$ and $(\mathbf{y}_1^{\mathbf{z}_0,\mathbf{x}'_1})_{\mathbf{z}_0 \in \{0,1\}^{\ell_1}}$ to S_1. Denote the output of R_0 by $\mathbf{w}_0$ and the output of R_1 by $\mathbf{w}_1$.
  6. *(Translating the shares)* R_0 samples the Σ_1-share $\mathbf{m}_0 \leftarrow \mathsf{LRRand}_{\Sigma_1}$, and R_1 samples the Σ_1-partial share $\mathbf{m}'_1 \leftarrow \mathsf{LRPartial}_{\Sigma_1}$. R_0 and R_1 call $\mathcal{F}_{\Sigma_1,\Sigma_2}^{\mathsf{translate}}$ with inputs $(\mathbf{w}_0, \mathbf{m}_0)$ to R_0 and $(\mathbf{w}_1, \mathbf{m}'_1)$ to R_1. Denote the output of R_1 by $\mathbf{m}_1$.
- **Outputs:** R_0 outputs $\mathbf{m}_0$ and R_1 outputs $\mathbf{m}_1$.

Figure 11: Protocol $\mathsf{LROLE}_{\Sigma_1,\Sigma_2}$

Let us denote by $\mathsf{OLE} : \{0,1\}^2 \to \{0,1\}$ to two-party OLE function, that takes from Alice a pair of bits (a, b) takes from Bob a bit x and returns $a \cdot x \oplus b$.

Lemma 5.2. *Let $2 \leq n \leq 4$ be the number of parties, let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be the parties, and let $\phi : \{S_0, S_1, R_0, R_1\} \to \mathcal{P}$ be a surjective role-assigning mapping. For $i \in [2]$, let Σ_i be a two-party (μ, ϵ_i)-local leakage resilient secret sharing scheme with share of length ℓ_i. Let $\mathsf{size}(\mathsf{refresh}), \mathsf{size}(\mathsf{matrix})$ and $\mathsf{size}(\mathsf{out})$ be the circuit size of the circuits f_{refresh}, f_{matrix} and f_{out}, respectively, as defined in $\mathsf{LROLE}_{\Sigma_1,\Sigma_2}$.*

Then the senders-receivers protocol $(\mathsf{LROLE}_{\Sigma_1,\Sigma_2}, \mathcal{P}, \phi)$ *computes the function* OLE *with strong* (μ', ϵ')*-composable leakage-resilient passive security with respect to* Σ_1*, where*

$$\mu' = \lfloor \mu/4 \rfloor ,$$
$$\epsilon' = (2 \cdot \mathsf{size}(\mathsf{refresh}))\epsilon_1 + (2 \cdot 2^{2\ell_1} \cdot \ell_1 + 2 \cdot \mathsf{size}(\mathsf{matrix}) + \mathsf{size}(\mathsf{out}))\epsilon_2.$$

The complexity of the protocol is polynomial in the complexity of Σ_1, Σ_2 *and in* 2^{ℓ_1}*.*

We prove Lemma 5.2 in Section B.2. The following remark explains how to instantiate the parameters in the lemma.

Remark 5.3 (Instantiation)**.** *For* $i \in \{1, 2\}$*, let* Σ_i *to be the* $(\mu_i = \lfloor \log(\kappa_i) \rfloor , \epsilon_i = 1/\kappa_i)$*-local leakage resilient secret sharing scheme promised by Theorem 4.8 instantiated with security parameter* κ_i*. We claim that there exist two polynomials,* p_1 *and* p_2*, such that by setting* $\kappa_i := p_i(\kappa)$ *we obtain* $\mu' = \lfloor \log(\kappa) \rfloor$ *and* $\epsilon' = 1/\kappa$*, as well as a polynomial complexity for the protocol.*

To see this, first observe that $\mathsf{size}(\mathsf{refresh}) = \mathrm{polylog}(\kappa_1)$*. Therefore, by taking* $\kappa_1 = p_1(\kappa)$ *for some polynomial* $p_1(\cdot)$*, we get that the parameters of* Σ_1 *satisfy*

$$\mu_1/4 \geq \lfloor \log(\kappa) \rfloor \quad \textit{and} \quad 2 \cdot \mathsf{size}(\mathsf{refresh})\epsilon_1 \leq \frac{1}{2\kappa}.$$

Next, letting $\ell_1 = \lceil 9 \log(\kappa_1) + 15 \rceil$ *be the share length of* Σ_1*, observe that* $\mathsf{size}(\mathsf{matrix}) = 2^{O(\log(\ell_1))} = \mathrm{poly}(\kappa_1)$ *and* $\mathsf{size}_{\mathsf{out}} = \mathrm{polylog}(\kappa_1, \kappa_2)$*. Therefore there is some polynomial* p_2 *such that for* $\kappa_2 = p_2(\kappa)$ *we get that the parameters of* Σ_2 *satisfy*

$$\mu_2/4 \geq \lfloor \log(\kappa) \rfloor \quad \textit{and} \quad (2 \cdot 2^{2\ell_1} \cdot \ell_1 + 2 \cdot \mathsf{size}(\mathsf{matrix}) + \mathsf{size}(\mathsf{out}))\epsilon_2 \leq \frac{1}{2\kappa}.$$

Overall, we obtained $\mu' = \lfloor \log(\kappa) \rfloor$ *and* $\epsilon' = 1/\kappa$ *and polynomial running time.*

5.2 The Compiler

In this section we present our final compiler. First, we need the following definition.

Definition 5.4. *A monotone formula* χ *is* admissible *if (1) it uses only AND and OR gates with fan-in two, (2) it uses no constants*[7]*, (3) The first layer of the circuit (whose incoming wires are all input variables) consists only of AND gates that both of their inputs are two distinct input variables. Further, input variables are not connected to any other gate. Accordingly, we refer to the first layer of AND gates as* input gates*.*

Jumping ahead, we will later see (Claim 7.4) that every threshold function can be realized by an explicit admissible monotone formula of logarithmic depth. We continue with the main theorem of this section.

Theorem 5.5. *There is a deterministic compiler* C *that takes as an input:*

- *an admissible monotone formula* $\chi : \{0,1\}^n \to \{0,1\}$ *whose depth is denoted by* d*;*
- *a function* $f : \{0,1\}^{\ell_{\mathsf{input}}} \to \{0,1\}^{\ell_{\mathsf{out}}}$*, represented as a Boolean circuit;*

[7] This is without loss of generality for every non-constant monotone formula

- *a statistical security parameter* 1^κ.

The compiler outputs:

- *a family* $\mathfrak{F}$ *of deterministic two-party solitary-output functionalities where each functionality in* $\mathfrak{F}$ *is represented by a Boolean circuit with input length* $O(\log(n \cdot 2^d \cdot |f| \cdot \kappa))$*, size* $\mathrm{polylog}(n, 2^d, |f|, \kappa)$ *and depth* $O(\log(n \cdot 2^d \cdot |f| \cdot \kappa))$. .
- *a description of an* n*-party protocol* π *in the* $\mathfrak{F}$*-hybrid model and an input encoder* Enc *(represented as a Boolean circuit) that compute* f *with* $(\chi, \lfloor \log(\kappa) \rfloor, 1/\kappa)$*-leakage-resilient dual security.*
- *A pair of circuits* $(\mathsf{Sim}, \mathsf{Ext})$ *that form the corresponding simulator and extractor for* (π, Enc).

The running time of C *is* $\mathrm{poly}(n, 2^d, |f|, \kappa)$*, where* $|f|$ *is the circuit size of* f *and* d *is the depth of the formula* χ*. Consequenlty, a similar bound applies to the complexity of* $(\pi, \mathsf{Enc}, \mathsf{Sim}, \mathsf{Ext})$.

Proof. Fix a function $f : \{0,1\}^{\ell_{\mathsf{input}}} \to \{0,1\}^{\ell_{\mathsf{out}}}$, an admissible monotone formula χ and a security parameter κ. Let m denote the number of input AND gates in χ. We represent χ by a triple (h, AND_2^m, T) as follows. Let $h : \{0,1\}^m \to \{0,1\}$ be the sub-formula of χ that consists of all the gates of χ except the input AND gates; let $\mathsf{AND}_2^m : \{0,1\}^n \to \{0,1\}^m$ be a single layer circuit that consists of m parallel calls to 2-ary AND (all the input gates of χ), and let $T : [m] \times [2] \to [n]$ be a mapping that maps an input wire into an input variable. That is, we identify each input wire by a pair $(i, b) \in [m] \times [2]$ where $i \in [m]$ specifies an input gate and $b \in [2]$ specifies whether it is the left or right input wire of the i-th input gate. To describe our construction, we first need the following theorem from [AK25].

Theorem 5.6 ([AK25])**.** *There is a deterministic compiler* C' *that takes as an input (1) a depth-*d *monotone formula* h *on* m *inputs and a single output bit, that uses only AND and OR gates with fan-in* 2*, and uses no constants, and (2) an* m*-party functionality* $\mathcal{F}$*, represented as a Boolean circuit. The compiler outputs an* m*-party protocol* π_h *in the* OLE*-hybrid model, that computes the functionality* $\mathcal{F}$ *with* h*-dual security. The compiler also outputs the description of the corresponding simulator* Sim_h *and the extractor* Ext_h*. The running time of* C' *is* $\mathrm{poly}(|\mathcal{F}| \cdot n \cdot 2^d)$*, and the running time of* Sim_h *and* Ext_h *and the number of instructions in* π_h *is bounded by* $|\mathcal{F}| \cdot \mathrm{poly}(n \cdot 2^d)$*, where* $|\mathcal{F}|$ *is the circuit size of* $\mathcal{F}$ *measured by the number of wires.*

Let C' be the compiler promised in Theorem 5.6, and let $\mathcal{F}$ be the m-party functionality that takes $\mathbf{x}^i \in \{0,1\}^{\ell_{\mathsf{input}}}$ from the i-th party, computes $\mathbf{x} = \mathbf{x}^1 \oplus \ldots \oplus \mathbf{x}^m$, and outputs $f(\mathbf{x})$ to all the parties. Let $(\pi_h, \mathsf{Sim}_h, \mathsf{Ext}_h) = C'(h, \mathcal{F})$, where π_h is an m-party protocol among the parties $w_1, \ldots, w_m$ that computes $\mathcal{F}$ with h-dual security. Let us set $\kappa' := \kappa \cdot |\pi_h|$, and let Σ_1 and Σ_2 be chosen as in Remark 5.3 with respect to the security parameter κ'. We denote the parameters of Σ_i by (μ_i, ϵ_i), and note that $\mu_i \geq \lfloor \log(\kappa) \rfloor$ and $\epsilon_i \leq \frac{1}{\kappa \cdot |\pi_h|}$.

Overview. We identify w_i, the i-th party in π_h, with the i-th input wire of h. Observe that w_i is the output wire of some AND gate g, say, the j-th input gate of χ. Let us denote by $\mathsf{Left}(w_i)$ (resp., $\mathsf{Right}(w_i)$) the index of the left (resp., right) input bit of g, i.e., $\mathsf{Left}(w_i) = T(j, 1)$ (resp., $\mathsf{Right}(w_i) = T(j, 2)$). At a high level, we construct a protocol π with n parties, $p_1, \ldots, p_n$, where the parties $p_{\mathsf{Left}(w_i)}$ and $p_{\mathsf{Right}(w_i)}$ emulate the execution of w_i in the protocol π_h. In particular, at every point in the execution, the parties $p_{\mathsf{Left}(w_i)}$ and $p_{\mathsf{Right}(w_i)}$ hold a leakage-resilient secret sharing of the view of w_i. For passive security we guarantee that if at most one of $p_{\mathsf{Left}(w_i)}$ and $p_{\mathsf{Right}(w_i)}$ is

passively corrupt then the view of w_i remains private. For active security, if both $p_{\mathsf{Left}(w_i)}$ and $p_{\mathsf{Right}(w_i)}$ are honest, then the emulated w_i is considered as an honest party, and otherwise as a corrupt party. We continue with a detailed description of our compiler.

The encoder. The encoder Enc takes the input $\mathbf{x} \in \{0,1\}^{\ell_{\mathsf{input}}}$, and samples an m-out-of-m secret sharing $(\mathbf{x}^1, \ldots, \mathbf{x}^m)$ of $\mathbf{x}$. For every $i \in [m]$ the encoder samples $(\vec{\mathbf{x}}_0^i, \vec{\mathbf{x}}_1^i) \leftarrow \mathsf{LRvShare}_{\Sigma_1}(\mathbf{x}^i)$ and sets $\vec{\mathbf{x}}_0^i$ as part of the input of $p_{\mathsf{Left}(w_i)}$, and $\vec{\mathbf{x}}_1^i$ as part of the input of $p_{\mathsf{Right}(w_i)}$. Observe that the j-th element of $\vec{\mathbf{x}}_0^i$ and $\vec{\mathbf{x}}_1^i$, denoted $\vec{\mathbf{x}}_0^i[j]$ and $\vec{\mathbf{x}}_1^i[j]$, form a Σ_1-sharing of the j-th bit of $\mathbf{x}^i$.

The protocol π. The protocol is defined as follows.

Protocol π

The protocol involves n parties $p_1, \ldots, p_n$.

- **Primitives:** The protocol requires the following primitives.
  - We let $(\vec{0}_{\mathsf{Left}}, \vec{0}_{\mathsf{Right}})$ and $(\vec{1}_{\mathsf{Left}}, \vec{1}_{\mathsf{Right}})$ be some arbitrary Σ_1-sharing of 0 and 1, respectively.
  - The sub-protocols: $\mathsf{LROp}_{\Sigma_1}(*)$ from Figure 5, $\mathsf{LROut}_{\Sigma_1}$ from Figure 7, $\mathsf{LRAbort}_{\Sigma_1}$ from Figure 8, and $\mathsf{LROLE}_{\Sigma_1,\Sigma_2}$ from Figure 11.
- **Inputs:** For every $i \in [\ell_{\mathsf{input}}]$, the parties $p_{\mathsf{Left}(w_i)}$ and $p_{\mathsf{Right}(w_i)}$ hold Σ_1-shares $\vec{\mathbf{x}}_0^i$ and $\vec{\mathbf{x}}_1^i$, respectively.
- **Computation:** The protocol follows the instructions of π_h so that for every party w in π_h, and every variable x^j in its view, the parties $p_{\mathsf{Left}(w)}$ and $p_{\mathsf{Right}(w)}$ hold Σ_1-shares of $\mathsf{val}(x^j)$ in the list of variables $X^{j,0}$ and $X^{j,1}$, respectively. For every $i = 1, \ldots, |\pi_h|$, the i-th statement d_i in π_h is compiled in π in the following way.
  - If the statement d_i is $\mathsf{input}(w_j, j', x^k)$ then we let $p_{\mathsf{Left}(w_j)}$ and $p_{\mathsf{Right}(w_j)}$ read from their input tape the Σ_1-shares of the j'-th input bit of w_j, that is, $\mathbf{x}_0^j[j']$ and $\mathbf{x}_1^j[j']$, respectively, and store it in the list of variables $X^{k,0}$ and $X^{k,1}$, respectively.
  - If the statement d_i is $\mathsf{transmit}(w_{j_1}, w_{j_2}, x^{k_1}, x^{k_2})$, then $p_{\mathsf{Left}(w_{j_1})}$ and $p_{\mathsf{Right}(w_{j_1})}$ hold Σ_1-shares of $\mathsf{val}(x^{k_1})$ in the lists of variables $X^{k_1,0}$ and $X^{k_1,1}$, respectively, and we denote the shares by $\mathbf{m}_0$ and $\mathbf{m}_1$. We let $p_{\mathsf{Left}(w_{j_1})}, p_{\mathsf{Right}(w_{j_1})}, p_{\mathsf{Left}(w_{j_2})}$, and $p_{\mathsf{Right}(w_{j_2})}$ execute an instance of $\mathsf{LROLE}_{\Sigma_1,\Sigma_2}$, where $p_{\mathsf{Left}(w_{j_1})}$ and $p_{\mathsf{Right}(w_{j_1})}$ act as senders with inputs $\mathbf{a}_0 = \mathbf{m}_0$, $\mathbf{b}_0 = \vec{0}_{\mathsf{Left}}$, $\mathbf{a}_1 = \mathbf{m}_1$ and $\mathbf{b}_1 = \vec{0}_{\mathsf{Left}}$, and $p_{\mathsf{Left}(w_{j_2})}$ and $p_{\mathsf{Right}(w_{j_2})}$ act as receivers with inputs $\mathbf{x}_0 = \vec{1}_{\mathsf{Left}}$ and $\mathbf{x}_1 = \vec{1}_{\mathsf{Right}}$. The outputs of the receivers $p_{\mathsf{Left}(w_{j_2})}$ and $p_{\mathsf{Right}(w_{j_2})}$ are stored in $X^{k_2,0}$ and $X^{k_2,1}$, respectively.
  - If the statement d_i is $\mathsf{comp}(w_j, *, (x^{k_1}, x^{k_2}), x^{k_3})$ for $* \in \{+, \times\}$, then $p_{\mathsf{Left}(w_j)}$ and $p_{\mathsf{Right}(w_j)}$ hold Σ_1-shares of $\mathsf{val}(x^{k_1})$ (resp., $\mathsf{val}(x^{k_2})$) in the lists of variables $X^{k_1,0}$ and $X^{k_1,1}$ (resp., $X^{k_2,0}$ and $X^{k_2,1}$). Denote the shares by $(\mathbf{v}_0^{k_1}, \mathbf{v}_1^{k_1})$ and $(\mathbf{v}_0^{k_2}, \mathbf{v}_1^{k_2})$. Then $p_{\mathsf{Left}(w_j)}$ and $p_{\mathsf{Right}(w_j)}$ execute $\mathsf{LROp}_{\Sigma_1}(*)$ with inputs $(\mathbf{v}_0^{k_1}, \mathbf{v}_1^{k_1})$ and $(\mathbf{v}_0^{k_2}, \mathbf{v}_1^{k_2})$, respectively, and the outputs $\mathbf{z}_0$ and $\mathbf{z}_1$ are stored in the lists of variables $X^{k_3,0}$ and $X^{k_3,1}$, respectively.
  - If the statement d_i is $\mathsf{comp}(w_j, \mathsf{rand}, \emptyset, x^k)$ then $p_{\mathsf{Left}(w_j)}$ samples $\mathbf{r}_0 \leftarrow \mathsf{LRRand}_{\Sigma_1}$ and $p_{\mathsf{Right}(w_j)}$ samples $\mathbf{r}_1 \leftarrow \mathsf{LRRand}_{\Sigma_1}$, and they store the shares in the list of variables $X^{k,0}$ and $X^{k,1}$, respectively.
  - If the statement d_i is $\mathsf{abort}(w_j, x^k)$, then $p_{\mathsf{Left}(w_j)}$ and $p_{\mathsf{Right}(w_j)}$ hold Σ_1-shares of $\mathsf{val}(x^k)$, denoted by $\mathbf{v}_0$ and $\mathbf{v}_1$. $p_{\mathsf{Left}(w_j)}$ and $p_{\mathsf{Right}(w_j)}$ execute $\mathsf{LRAbort}_{\Sigma_1}$ with inputs $\mathbf{v}_0$ and $\mathbf{v}_1$.

– If the statement d_i is $\mathsf{func}(\mathsf{OLE}, w_{j_1}, w_{j_2}, (x^{k_1}, x^{k_2}), x^{k_3}, x^{k_4})$, then $p_{\mathsf{Left}(w_{j_1})}, p_{\mathsf{Right}(w_{j_1})}$ hold Σ_1-shares of $\mathsf{val}(x^{k_1})$ and $\mathsf{val}(x^{k_2})$ in the lists of variables $(X^{k_1,0}, X^{k_1,1})$ and $(X^{k_2,0}, X^{k_2,1})$. Similarly, $p_{\mathsf{Left}(w_{j_2})}, p_{\mathsf{Right}(w_{j_2})}$ hold Σ_1-shares of $\mathsf{val}(x^{k_3})$ in the lists of variables $(X^{k_3,0}, X^{k_3,1})$. Denote the Σ_1-shares by $(\mathbf{a}_0, \mathbf{a}_1)$, $(\mathbf{b}_0, \mathbf{b}_1)$ and $(\mathbf{x}_0, \mathbf{x}_1)$, respectively. The parties execute $\mathsf{LROLE}_{\Sigma_1,\Sigma_2}$ where the senders $p_{\mathsf{Left}(w_{j_1})}, p_{\mathsf{Right}(w_{j_1})}$ input $(\mathbf{a}_0, \mathbf{b}_0)$ and $(\mathbf{a}_1, \mathbf{b}_1)$, respectively, and the receivers $p_{\mathsf{Left}(w_{j_2})}, p_{\mathsf{Right}(w_{j_2})}$ input $\mathbf{x}_0$ and $\mathbf{x}_1$, respectively. Denote the outputs of $p_{\mathsf{Left}(w_{j_2})}, p_{\mathsf{Right}(w_{j_2})}$ by $\mathbf{z}_0$ and $\mathbf{z}_1$. The outputs are stored in $X^{k_4,0}$ and $X^{k_4,1}$, respectively.

– If the statement d_i is $\mathsf{out}(w, j, x^k)$ then $p_{\mathsf{Left}(w)}, p_{\mathsf{Right}(w)}$ hold Σ_1-shares of $\mathsf{val}(x^k)$ in the lists of variables $(X^{k,0}, X^{k,1})$, and denote the Σ_1-shares by $\mathbf{v}_0$ and $\mathbf{v}_1$, respectively. $p_{\mathsf{Left}(w)}, p_{\mathsf{Right}(w)}$ execute $\mathsf{LROut}_{\Sigma_1}$ with $\mathbf{v}_0$ and $\mathbf{v}_1$ as inputs to obtain a bit b and set the j-th output bit to be b.

Figure 12: Protocol π

The simulator. We continue with a formal description of the simulator.

Simulator Sim

- **Primitives:** The protocol requires the following primitives.
  - We let $(\vec{0}_{\mathsf{Left}}, \vec{0}_{\mathsf{Right}})$ and $(\vec{1}_{\mathsf{Left}}, \vec{1}_{\mathsf{Right}})$ be some arbitrary Σ_1-sharing of 0 and 1, respectively.
  - The simulator Sim_h of π_h.
  - The sub-protocol $\mathsf{LROp}_{\Sigma_1}(*)$ from Figure 5 and the corresponding simulator $\mathsf{Sim}_{\mathsf{LROp}(*)}$.
  - The sub-protocol $\mathsf{LROut}_{\Sigma_1}$ from Figure 7 and the corresponding simulator $\mathsf{Sim}_{\mathsf{LROut}}$.
  - The sub-protocol $\mathsf{LRAbort}_{\Sigma_1}$ from Figure 8 and the corresponding simulator $\mathsf{Sim}_{\mathsf{LRAbort}}$.
  - The sub-protocol $\mathsf{LROLE}_{\Sigma_1,\Sigma_2}$ from Figure 11 and the corresponding simulator $\mathsf{Sim}_{\mathsf{LROLE}}$.
- **Inputs:** The simulator takes as an input a subset $Z \subseteq \{p_1, \ldots, p_n\}$, and an output $\mathbf{y} \in \{0,1\}^{\ell_{\mathsf{out}}}$.
- **Fake input initialization:** For every w_i the simulator samples $\mathbf{x}^i \leftarrow \{0,1\}^{\ell_{\mathsf{input}}}$, and $(\vec{\mathbf{x}}_0^i, \vec{\mathbf{x}}_1^i) \leftarrow \mathsf{LRvShare}_{\Sigma_1}(\mathbf{x}^i)$.
- **Fake views for Z_h.** Let $Z_h := \{w_i : p_{\mathsf{Left}(w_i)} \in Z \wedge p_{\mathsf{Right}(w_i)} \in Z\}$ be the subset of $\{w_1, \ldots, w_m\}$ containing every w_i with $p_{\mathsf{Left}(w_i)} \in Z$ and $p_{\mathsf{Right}(w_i)} \in Z$. The simulator executes $\mathsf{Sim}_h(Z_h, (\mathbf{x}^i)_{w_i \in Z_h}, \mathbf{y})$ to obtain the views $\mathcal{V}' = (\mathsf{view}'_i)_{w_i \in Z_h}$ that correspond to the protocol π_h.
- **Protocol Simulation:** The simulator then initializes empty views $\mathcal{V} := (\mathsf{view}_1, \ldots, \mathsf{view}_n)$. For every $i = 1, \ldots, |\pi_h|$, and for the i-th statement d_i in π_h, the simulator updates $\mathcal{V}$ in the following way.
  - If the statement d_i is $\mathsf{input}(w_j, j', x^k)$, then denote the j'-th element of $\vec{\mathbf{x}}_0^j$ and $\vec{\mathbf{x}}_1^j$ by $\mathbf{x}_0 := \mathbf{x}_0^j[j']$ and $\mathbf{x}_1 := \mathbf{x}_1^j[j']$, respectively. The simulator assigns $\mathbf{x}_0$ to $X^{k,0}$ and $\mathbf{x}_1$ to $X^{k,1}$, and updates $\mathcal{V}$ accordingly.
  - If the statement d_i is $\mathsf{transmit}(w_{j_1}, w_{j_2}, x^{k_1}, x^{k_2})$, then denote the values of the lists of variables $X^{k_1,0}$ and $X^{k_1,1}$ according to the views of $p_{\mathsf{Left}(w_{j_1})}$ and $p_{\mathsf{Right}(w_{j_1})}$ by $\mathbf{a}_0$ and $\mathbf{a}_1$. The simulation of $\mathsf{LROLE}_{\Sigma_1,\Sigma_2}$ is done according to one of the following 4 cases.
    * If $w_{j_1}, w_{j_2} \in Z_h$ then the simulation is done by executing $\mathsf{LROLE}_{\Sigma_1,\Sigma_2}$ with $p_{\mathsf{Left}(w_{j_1})}$ and $p_{\mathsf{Right}(w_{j_1})}$ as senders with inputs $(\mathbf{a}_0, \vec{0}_{\mathsf{Left}})$ and $(\mathbf{a}_1, \vec{0}_{\mathsf{Right}})$, and with $p_{\mathsf{Left}(w_{j_2})}$ and $p_{\mathsf{Right}(w_{j_2})}$ as receivers with inputs $\vec{1}_{\mathsf{Left}}$ and $\vec{1}_{\mathsf{Right}}$.

* If $w_{j_1} \in Z_h$ and $w_{j_2} \notin Z_h$ then the simulator samples $(\mathsf{out}_0, \mathsf{out}_1) \leftarrow \mathsf{LRShare}_{\Sigma_1}(0)$ and the simulation is done by executing $\mathsf{Sim}_{\mathsf{LROLE}}(Z \cap \left\{p_{\mathsf{Left}(w_{j_1})}, p_{\mathsf{Right}(w_{j_1})}, p_{\mathsf{Left}(w_{j_2})}, p_{\mathsf{Right}(w_{j_2})}\right\}, 0, \mathsf{aux})$, where aux contains $(\mathbf{a}_0, \vec{0}_{\mathsf{Left}})$ and $(\mathbf{a}_1, \vec{0}_{\mathsf{Right}})$.
* If $w_{j_2} \in Z_h$ and $w_{j_1} \notin Z_h$ then the simulator holds $\mathsf{val}(x^{k_2})$ as part of view'_{j_2}. The simulator samples $(\mathsf{out}_0, \mathsf{out}_1) \leftarrow \mathsf{LRShare}_{\Sigma_1}(\mathsf{val}(x^{k_2}))$, and the simulation is done by executing $\mathsf{Sim}_{\mathsf{LROLE}}(Z \cap \left\{p_{\mathsf{Left}(w_{j_1})}, p_{\mathsf{Right}(w_{j_1})}, p_{\mathsf{Left}(w_{j_2})}, p_{\mathsf{Right}(w_{j_2})}\right\}, \mathsf{val}(x^{k_2}), \mathsf{aux})$, where aux contains $(\vec{1}_{\mathsf{Left}}, \vec{1}_{\mathsf{Right}})$.
* If $w_{j_1}, w_{j_2} \notin Z_h$. The simulator samples $(\mathsf{out}_0, \mathsf{out}_1) \leftarrow \mathsf{LRShare}_{\Sigma_1}(0)$ and the simulation is done by executing $\mathsf{Sim}_{\mathsf{LROLE}}(Z \cap \left\{p_{\mathsf{Left}(w_{j_1})}, p_{\mathsf{Right}(w_{j_1})}, p_{\mathsf{Left}(w_{j_2})}, p_{\mathsf{Right}(w_{j_2})}\right\}, 0, \emptyset)$.

– If the statement d_i is $\mathsf{comp}(w_j, *, (x^{k_1}, x^{k_2}), x^{k_3})$ for $* \in \{+, \times\}$, then denote the values of the lists of variables $X^{k_1,0}$ and $X^{k_1,1}$ (resp., $X^{k_2,0}$ and $X^{k_2,1}$) according to the views of $p_{\mathsf{Left}(w_j)}$ and $p_{\mathsf{Right}(w_j)}$ by $(\mathbf{v}_0^{k_1}, \mathbf{v}_1^{k_1})$ (resp., $(\mathbf{v}_0^{k_2}, \mathbf{v}_1^{k_2})$).
* If $w_j \in Z_h$ then the simulation is done by executing $\mathsf{LROp}_{\Sigma_1}(*)$ with $p_{\mathsf{Left}_{w_j}}$ and $p_{\mathsf{Right}_{w_j}}$ with inputs $(\mathbf{v}_0^{k_1}, \mathbf{v}_0^{k_2})$ and $(\mathbf{v}_1^{k_1}, \mathbf{v}_1^{k_2})$.
* Otherwise $w_j \notin Z_h$. In this case the simulation is done by sampling $(\mathsf{out}_0, \mathsf{out}_1) \leftarrow \mathsf{LRShare}_{\Sigma_1}(0)$ and executing $\mathsf{Sim}_{\mathsf{LROp}(*)}(Z \cap \left\{p_{\mathsf{Left}_{w_j}}, p_{\mathsf{Right}_{w_j}}\right\}, (\mathsf{out}_0, \mathsf{out}_1), \emptyset)$.

– If the statement d_i is $\mathsf{comp}(w_j, \mathsf{rand}, \emptyset, x^k)$ then we do as follows.
* If $w_j \in Z_h$ then the simulator holds $\mathsf{val}(x^k)$ as part of view'_j. The simulator samples $(\mathbf{r}_0, \mathbf{r}_1) \leftarrow \mathsf{LRShare}_{\Sigma_1}(\mathsf{val}(x^k))$ and stores the shares at $X^{k,0}$ and $X^{k,1}$, respectively.
* Otherwise, if $w_j \notin Z_h$ then the simulator samples $(\mathbf{r}_0, \mathbf{r}_1) \leftarrow \mathsf{LRShare}_{\Sigma_1}(0)$ and stores the shares at $X^{k,0}$ and $X^{k,1}$, respectively.

– If the statement d_i is $\mathsf{abort}(w_j, x^k)$, then $p_{\mathsf{Left}(w_j)}$ and $p_{\mathsf{Right}(w_j)}$ then the simulation of $\mathsf{LRAbort}_{\Sigma_1}$ is done by setting the output of $\mathcal{F}_{\Sigma_1}^{\mathsf{out}}$ to be 0.

– If the statement d_i is $\mathsf{func}(\mathsf{OLE}, w_{j_1}, w_{j_2}, (x^{k_1}, x^{k_2}), x^{k_3}, x^{k_4})$, then denote the values of the lists of variables $X^{k_1,0}$ and $X^{k_1,1}$ according to the views of $p_{\mathsf{Left}(w_{j_1})}$ and $p_{\mathsf{Right}(w_{j_1})}$ by $\mathbf{a}_0$ and $\mathbf{a}_1$, the values of the lists of variables $X^{k_2,0}$ and $X^{k_2,1}$ according to the views of $p_{\mathsf{Left}(w_{j_1})}$ and $p_{\mathsf{Right}(w_{j_1})}$ by $\mathbf{b}_0$ and $\mathbf{b}_1$, and the values of the lists of variables $X^{k_3,0}$ and $X^{k_3,1}$ according to the views of $p_{\mathsf{Left}(w_{j_2})}$ and $p_{\mathsf{Right}(w_{j_2})}$ by $\mathbf{x}_0$ and $\mathbf{x}_1$. The simulation of $\mathsf{LROLE}_{\Sigma_1,\Sigma_2}$ is done according to one of the following 4 cases.
* If $w_{j_1}, w_{j_2} \in Z_h$ then the simulation is done by executing $\mathsf{LROLE}_{\Sigma_1,\Sigma_2}$ with $p_{\mathsf{Left}(w_{j_1})}$ and $p_{\mathsf{Right}(w_{j_1})}$ as senders with inputs $(\mathbf{a}_0, \mathbf{b}_0)$ and $(\mathbf{a}_1, \mathbf{b}_1)$, and with $p_{\mathsf{Left}(w_{j_2})}$ and $p_{\mathsf{Right}(w_{j_2})}$ as receivers with inputs $\mathbf{x}_0$ and $\mathbf{x}_1$.
* If $w_{j_1} \in Z_h$ and $w_{j_2} \notin Z_h$ then the simulator samples $(\mathsf{out}_0, \mathsf{out}_1) \leftarrow \mathsf{LRShare}_{\Sigma_1}(0)$ and the simulation is done by executing $\mathsf{Sim}_{\mathsf{LROLE}}(Z \cap \left\{p_{\mathsf{Left}(w_{j_1})}, p_{\mathsf{Right}(w_{j_1})}, p_{\mathsf{Left}(w_{j_2})}, p_{\mathsf{Right}(w_{j_2})}\right\}, 0, \mathsf{aux})$, where aux contains $(\mathbf{a}_0, \mathbf{b}_0)$ and $(\mathbf{a}_1, \mathbf{b}_1)$.
* If $w_{j_2} \in Z_h$ and $w_{j_1} \notin Z_h$ then the simulator holds $\mathsf{val}(x^{k_2})$ as part of view'_{j_2}. The simulator samples $(\mathsf{out}_0, \mathsf{out}_1) \leftarrow \mathsf{LRShare}_{\Sigma_1}(\mathsf{val}(x^{k_2}))$, and the simulation is done by executing $\mathsf{Sim}_{\mathsf{LROLE}}(Z \cap \left\{p_{\mathsf{Left}(w_{j_1})}, p_{\mathsf{Right}(w_{j_1})}, p_{\mathsf{Left}(w_{j_2})}, p_{\mathsf{Right}(w_{j_2})}\right\}, \mathsf{val}(x^{k_2}), \mathsf{aux})$, where aux contains $(\mathbf{x}_0, \mathbf{x}_1)$.
* If $w_{j_1}, w_{j_2} \notin Z_h$. The simulator samples $(\mathsf{out}_0, \mathsf{out}_1) \leftarrow \mathsf{LRShare}_{\Sigma_1}(0)$ and the simulation is done by executing $\mathsf{Sim}_{\mathsf{LROLE}}(Z \cap \left\{p_{\mathsf{Left}(w_{j_1})}, p_{\mathsf{Right}(w_{j_1})}, p_{\mathsf{Left}(w_{j_2})}, p_{\mathsf{Right}(w_{j_2})}\right\}, 0, \emptyset)$.

- If the statement d_i is $\mathsf{out}(w, j, x^k)$ then $p_{\mathsf{Left}(w)}, p_{\mathsf{Right}(w)}$ then then the simulation of $\mathsf{LROut}_{\Sigma_1}$ is done by setting the output of $\mathcal{F}^{\mathsf{out}}_{\Sigma_1}$ and the message from $p_{\mathsf{Right}(w_j)}$ to $p_{\mathsf{Left}(w_j)}$ to be y_j.

- **Output:** The simulator outputs $\mathsf{view}_1, \ldots, \mathsf{view}_n$.

Figure 13: Simulator Sim

The extractor. We continue with the description of the extractor.

Extractor Ext

- **Primitives:** Let Ext_h be the extractor of π_h.
- **Inputs:** The extractor takes as an input a set $S \subseteq \{p_1, \ldots, p_n\}$ that satisfies $\chi(S) = 1$, and views $(\mathsf{view}_i)_{p_i \in S}$.
- **Computation:** Let $S_h := \left\{w_i : p_{\mathsf{Left}(w_i)}, p_{\mathsf{Right}(w_i)} \in S\right\}$. First, for every $w_i \in S_h$, we recover a view view'_i as follows: for every variable x^k that belongs to w_i, let $\mathbf{v}_0$ (resp., $\mathbf{v}_1$) be the Σ_1-share corresponding to the value of $X^{k,0}$ (resp., $X^{k,1}$) from $\mathsf{view}_{\mathsf{Left}(w_i)}$ (resp., $\mathsf{view}_{\mathsf{Right}(w_i)}$), define the value of x^k to be $\mathsf{LRRecover}_{\Sigma_1}(\mathbf{v}_0, \mathbf{v}_1)$. Next, we execute $\mathsf{Ext}_h(S_h, (\mathsf{view}'_i)_{w_i \in S_h})$ to obtain an assignment $(\mathbf{x}^1, \ldots, \mathbf{x}^m)$, and set $\mathbf{x} := \mathbf{x}^1 \oplus \ldots \oplus \mathbf{x}^m$.
- **Output:** The extractor outputs $\mathbf{x}$.

Figure 14: Extractor Ext

It suffices to prove the following lemmas.

Lemma 5.7. *The protocol* (π, Enc) *computes* f *with* $(\chi, \lfloor \log(\kappa) \rfloor, 1/\kappa)$*-leakage resilient passive security, with respect to the simulator* Sim*.*

Lemma 5.8. *The protocol* (π, Enc) *computes* f *with* χ*-active correctness with abort, with respect to the extractor* Ext*.*

We prove Lemma 5.7 in Section B.3, and Lemma 5.8 in Section B.4. □

6 Leakage Resilient MPC: Pairwise Correlations

In this section we explain how to transform a leakage-resilient protocol with calls to two-party solitary-output functionalities into a protocol with pairwise correlated randomness and no calls to an ideal functionality. The overhead of the transformation grows polynomially with the circuit-size of the ideal functionality and exponentially with its depth. To maintain polynomial overhead, we will therefore use only "simple" functionalities that can be computed with polynomial size and logarithmic depth.

Our main building block, private simultaneous message protocol, is presented in Section 6.1. A general compiler is then presented in Section 6.2, and in Section 6.3 we apply the compiler on our protocol from Section 5.2 to obtain a leakage-resilient protocol in the pairwise correlations model.

6.1 Private Simultaneous Message Protocols

In private simultaneous message (PSM) protocols for a function f with ℓ input arguments, there are ℓ servers $Q_1, \ldots, Q_\ell$, each with a private input x_i, that share a common random string r. The protocol allows each server Q_i to send a single message $m_i := \mathsf{psm}_i(x_i; r)$ that depends only on its private input x_i and shared randomness r, such that the messages $(m_1, \ldots, m_\ell)$ reveal the output $f(x_1, \ldots, x_\ell)$, but no other information about the inputs $x_1, \ldots, x_\ell$. (Such protocols are also known as fully-decomposable randomized encoding; see [Ish13, App17].)

Definition 6.1 (PSM Protocols [FKN94])**.** *Let $X_1, \ldots, X_\ell, Z$ be finite sets, and let $X = X_1 \times \ldots \times X_\ell$. An ℓ-party PSM protocol* psm*, computing a ℓ-argument function $f : X \to Z$ consists of:*

- *A message computation function $\mathsf{psm}_i : X_i \times R \to M_i$, for every party $i \in \{1, \ldots, \ell\}$, where R is a finite set of common random inputs and M_i is a finite message domain.*
- *A reconstruction function $\mathsf{rec} : M_1 \times \ldots \times M_\ell \to Z$ that will be computed by the receiver.*

The protocol $\mathsf{psm} = \big(\mathsf{psm}_1, \ldots, \mathsf{psm}_\ell, \mathsf{rec}\big)$ should satisfy the following properties.

1. **(Correctness)** *For every $(x_1, \ldots, x_\ell) \in X$ and $r \in R$, $\mathsf{rec}(\mathsf{psm}_1(x_1; r), \ldots, \mathsf{psm}_m(x_\ell; r)) = f(x_1, \ldots, x_\ell)$.*
2. **(Privacy)** *There exists a simulator $\mathsf{Sim}_{\mathsf{psm}}$, such that for every $(x_1, \ldots, x_\ell) \in X$,*

$$\mathsf{Sim}_{\mathsf{psm}}\big(f(x_1, \ldots, x_\ell)\big) \equiv (\mathsf{psm}_1(x_1; r), \ldots, \mathsf{psm}_m(x_\ell; r)),$$

   for $r \leftarrow R$.

Note that PSM security addresses only the case where the parties $Q_1, \ldots, Q_\ell$ are honest.

Lemma 6.2 (Polynomial-time PSM Protocols [IK02])**.** *For every ℓ-argument functionality f that admits a Boolean $\mathbf{NC}^1$ circuit of size s, there exists a PSM protocol with complexity of $\mathrm{poly}(s)$. In particular, if $s = \mathrm{poly}(\ell)$, then there exists a PSM protocol with complexity $\mathrm{poly}(\ell)$.*

6.2 Our Construction

In this section, we generically transform leakage-resilient protocols that employ two-party solitary-output ideal functionalities into a leakage-resilient protocols with correlated randomness that uses no ideal functionalities.

Protocol overview. The high level idea is to replace every call of some parties p and q to an ideal functionality $\mathcal{F}$ with a two-party leakage-resilient secure computation in the correlated randomness model. To do so, we let the correlated randomness sample all the possible PSM messages of $\mathcal{F}$ and share them among p and q using leakage-resilient secret sharing. Let us denote by $(\mathbf{s}^0_{i,0}, \mathbf{s}^0_{i,1})$ and $(\mathbf{s}^1_{i,0}, \mathbf{s}^1_{i,1})$ the shares that correspond to the i-th PSM message when the i-th input bit is 0 and 1, respectively. Then the generator samples a random permutation $\sigma_i \in \{0, 1\}$, and provides the permuted shares $(\mathbf{s}^{\sigma_i}_{i,0}, \mathbf{s}^{1\oplus\sigma_i}_{i,0})$ to p and $(\mathbf{s}^{\sigma_i}_{i,1}, \mathbf{s}^{1\oplus\sigma_i}_{i,1})$ to q. In addition, if the i-th bit belongs to p then

p also receives σ_i, and otherwise q receives σ_i, i.e., the party that holds the i-th bit also learns the permutation.[8]

Consider the computation of $\mathcal{F}$ on input $\mathbf{x}$ where the output should be given to q. In this case, if p holds the i-th bit of $\mathbf{x}$ then we let p send $x_i \oplus \sigma_i$ to q, together with the corresponding share $\mathbf{s}_{i,0}^{x_i}$. In this way, q knows the location of the correct share she holds, i.e., $\mathbf{s}_{i,1}^{x_i}$ but learns nothing about the input bit x_i since σ_i is used as a one-time pad. Similarly, if the i-th bit belongs to q then q already holds σ_i and $\mathbf{s}_{i,1}^{x_i}$, so she only needs to learn $\mathbf{s}_{i,0}^{x_i}$. Therefore, we let q send $x_i \oplus \sigma_i$ to p, so p can respond with the correct share $\mathbf{s}_{i,0}^{x_i}$. Given all the shares q can recover all the PSM messages that correspond to $\mathbf{x}$ and thus the recover the output. We note that the perfect security of the PSM guarantees that q learns no information other than the output.

Theorem 6.3. *Let Σ be a two-party (μ, ϵ_Σ)-local-leakage-resilient secret sharing scheme with complexity t. Let $f : \{0,1\}^{\ell_{\mathsf{input}}} \to \{0,1\}^{\ell_{\mathsf{out}}}$ be a circuit, and let $\mathcal{F}_1, \ldots, \mathcal{F}_q$ be two-party solitary-output functionalities that admit a PSM of complexity s. Let $\Pi' = (\pi', \mathsf{Enc}')$ be a protocol that computes f with (χ, μ, ϵ')-leakage-resilient passive security in the $(\mathcal{F}_1, \ldots, \mathcal{F}_q)$-hybrid model. Then there exists a protocol $\Pi = (\pi, \mathsf{Enc}, \mathsf{Rand})$ that computes f with (χ, μ, ϵ)-leakage-resilient passive security with pairwise correlated randomness (and no calls to an ideal functionality), where*

$$\epsilon = \epsilon' + \left|\pi'\right| \cdot s \cdot \epsilon_\Sigma.$$

The complexity of π is $\mathrm{poly}(t, s, |\pi'|)$.

Proof. In the following, we denote the parties by $\mathcal{P} = \{p_1, \ldots, p_n\}$. To simplify notation, let us assume without loss of generality that every $\mathcal{F}_i$ takes ℓ bits from Alice and ℓ input bits from Bob, and returns ℓ' output bits to Bob. (We can always pad the input and output with zeroes.) Throughout, we assume that the first ℓ bits belong to Alice, and the last ℓ bits belong to Bob. Let $\mathsf{psm}^i = (\mathsf{psm}_1^i, \ldots, \mathsf{psm}_{2\ell}^i, \mathsf{rec})$ be the PSM scheme for $\mathcal{F}_i$ of complexity s guaranteed by Lemma 6.2. We assume without loss of generality that all PSM messages have the same length ℓ_{psm}.

The input encoder Enc**.** The input encoder Enc is identical to the input encoder Enc′.

The correlated randomness generator Rand**.** For every $\mathcal{F}_i$-statement $\mathsf{func}(\mathcal{F}_i, p, q, X, Y, Z)$ in π', where p and q are two parties and X, Y and Z are tuples of variables we let Rand generate correlated randomness between p and q in the following way.

Correlated Randomness for $\mathcal{F}_i$-Statement

Rand samples a random string $\mathbf{r}$ for psm^i, computes $\mathbf{m}_{j,b} := \mathsf{psm}_j^i(b; \mathbf{r})$ for every $j \in [2\ell]$ and $b \in \{0,1\}$, and samples shares of each message $(M_{j,b,0}, M_{j,b,1}) \leftarrow \mathsf{LRvShare}_\Sigma(\mathbf{m}_{j,b})$. In addition, Rand samples 2ℓ bits $(\sigma_1, \ldots, \sigma_{2\ell}) \leftarrow \{0,1\}^{2\ell}$ where every σ_i will be used as a permutation bit.

Finally, Rand outputs to p the shares that correspond to p's inputs $(M_{j,0,0}, M_{j,1,0})_{j\in[\ell]}$, the permuted shares that correspond to q's inputs $(M_{j,0\oplus\sigma_j,0}, M_{j,1\oplus\sigma_j,0})_{j\in\{\ell+1,\ldots,2\ell\}}$, and the permutations that correspond to p's inputs $(\sigma_j)_{j\in[\ell]}$. In addition, Rand outputs to q the permuted messages that correspond to p's inputs $(M_{j,0\oplus\sigma_j,1}, M_{j,1\oplus\sigma_j,1})_{j\in[\ell]}$, the (non-permuted) messages that correspond to q's inputs

[8] To simplify notation, in the actual protocol we simply give the party that holds the i-th bit the shares without permutation.

$(M_{j,0,1}, M_{j,1,1})_{j\in\{\ell+1,\ldots,2\ell\}}$, and the permutations that correspond to q's inputs $(\sigma_j)_{j\in\{\ell+1,\ldots,2\ell\}}$.

Figure 15: Correlated Randomness for $\mathcal{F}_i$-Statement

The protocol π. The protocol π is obtained from π' by replacing every $\mathcal{F}_i$-statement $\mathsf{func}(\mathcal{F}_i, p, q, X, Y, Z)$ statement in π with the following sub-protocol.

Sub-Protocol for $\mathcal{F}_i$-Statement

1. Let us denote the inputs of p and q to $\mathcal{F}_i$ by $\mathbf{w}$ and $\mathbf{y}$, respectively, and denote their j-th bit by w_j and y_j. We denote the correlated randomness corresponding to this statement by $(M_{j,b,0})_{j\in[\ell],b\in\{0,1\}}, (M_{j,b\oplus\sigma_j,0})_{j\in\{\ell+1,\ldots,2\ell\},b\in\{0,1\}}, (\sigma_j)_{j\in[\ell]}$ for p and $(M_{j,b\oplus\sigma_j,1})_{j\in[\ell],b\in\{0,1\}}, (M_{j,b,1})_{j\in\{\ell+1,\ldots,2\ell\},b\in\{0,1\}}, (\sigma_j)_{j\in\{\ell+1,\ldots,2\ell\}}$ for q.
2. For every $j \in [\ell]$, the party p sends $(M_{j,w_j,0}, \sigma_j \oplus w_j)_{j\in[\ell]}$ to q. Denote the message that q received by $(A_{j,0}, \sigma'_j)$.
3. For every $j \in \{\ell+1,\ldots,2\ell\}$, the party q computes $\sigma'_j := y_{j-\ell} \oplus \sigma_j$ and sends σ'_j to p.
4. Recall that for every $j \in \{\ell+1,\ldots,2\ell\}$ the party p holds a permuted pair $(M_{j,0\oplus\sigma_j,0}, M_{j,1\oplus\sigma_j,0})$. Denote by $A_{j,0}$ the element from the pair at location σ'_j (that is, $A_{j,0} := M_{j,0\oplus\sigma_j,0}$ if $\sigma'_j = 0$, and $A_{j,0} := M_{j,1\oplus\sigma_j,0}$ if $\sigma'_j = 1$). The party p sends $(A_{j,0})_{j\in\{\ell+1,\ldots,2\ell\}}$ to q.
5. Recall that for every $j \in [\ell]$, the party q holds a permuted pair $(M_{j,0\oplus\sigma_j,1}, M_{j,1\oplus\sigma_j,1})$. Denote by $A_{j,1}$ the element from the pair at location σ'_j (that is, $A_{j,1} := M_{j,0\oplus\sigma_j,1}$ if $\sigma'_j = 0$, and $A_{j,1} := M_{j,1\oplus\sigma_j,1}$ otherwise). In addition, for every $j \in \{\ell+1,\ldots,2\ell\}$ we let q set $A_{j,1} := M_{j,y_{j-\ell},1}$.

   For every $j \in [2\ell]$, party q recovers $\mathbf{m}_j := \mathsf{LRvRecover}_\Sigma(A_{j,0}, A_{j,1})$. Then, q computes $\mathbf{z} := \mathsf{rec}(\mathbf{m}_1,\ldots,\mathbf{m}_{2\ell})$, and stores the output $\mathbf{z}$ at the variables Z.

Figure 16: Sub-Protocol for $\mathcal{F}_i$-Statement

The simulator Sim**.** Let Sim′ be the simulator of π'. On input $(Z, f(\mathbf{x}))$ the simulator Sim samples $(\mathsf{view}_1,\ldots,\mathsf{view}_n) \leftarrow \mathsf{Sim}'(Z, f(\mathbf{x}))$. Then, for every $\mathcal{F}_i$-statement $\mathsf{func}(\mathcal{F}_i, p, q, X, Y, Z)$, the simulator Sim updates the views corresponding to p and q by simulating the sub-protocol for $\mathcal{F}_i$ in the following way.

Update Views for $\mathcal{F}_i$-Statement

Let us denote the inputs of p and q to $\mathcal{F}_i$ according to view_p and view_q by $\mathbf{w}$ and $\mathbf{y}$, respectively. In addition, we denote the output of q according to view_q by $\mathbf{z}$. (Observe that, since view_p and view_q are the views generated by Sim′, it is possible that $\mathcal{F}_i(\mathbf{w},\mathbf{y}) \neq \mathbf{z}$.)

Case 1: $p, q \in Z$**.** First, we consider the case where $p, q \in Z$. In this case, the simulator Sim samples the correlated randomness $(M_{j,b,0})_{j\in[\ell],b\in\{0,1\}}, (M_{j,b\oplus\sigma_j,0})_{j\in\{\ell+1,\ldots,2\ell\},b\in\{0,1\}}, (\sigma_j)_{j\in[\ell]}$ for p and $(M_{j,b\oplus\sigma_j,1})_{j\in[\ell],b\in\{0,1\}}, (M_{j,b,1})_{j\in\{\ell+1,\ldots,2\ell\},b\in\{0,1\}}, (\sigma_j)_{j\in\{\ell+1,\ldots,2\ell\}}$ for q, as described in Figure 15. Then, Sim updates the views of p and q by executing the sub-protocol from Figure 16 with the correlated randomness it sampled and inputs $\mathbf{w}$ and $\mathbf{y}$. This concludes the update procedure for the case where $p, q \in Z$.

Case 2: $p \notin Z$ or $q \notin Z$. Assume that either $p \notin Z$ or $q \notin Z$. Let $\mathsf{Sim}_{\mathsf{psm}}$ be the simulator of psm^i. The simulator Sim samples $(\mathbf{m}_1, \ldots, \mathbf{m}_{2\ell}) \leftarrow \mathsf{Sim}_{\mathsf{psm}}(\mathbf{z})$, and for every $j \in [2\ell]$ the simulator samples $(A_{j,0}, A_{j,1}) \leftarrow \mathsf{LRvShare}(\mathbf{m}_j)$ and $(B_{j,0}, B_{j,1}) \leftarrow \mathsf{LRvShare}(\vec{0})$, where $\vec{0}$ is a vector of ℓ_{psm} zeros. The simulator also samples random bits $\sigma_1, \ldots, \sigma_{2\ell}$.

For every $j \in [\ell]$ the simulator sets $M_{j,w_j,0} := A_{j,0}$, $M_{j,w_j,1} := A_{j,1}$, $M_{j,1\oplus w_j,0} := B_{j,0}$, and $M_{j,1\oplus w_j,1} := B_{j,1}$. For every $j \in \{\ell+1, \ldots, 2\ell\}$ the simulator sets $M_{j,y_{j-\ell},0} := A_{j,0}$, $M_{j,y_{j-\ell},1} := A_{j,1}$, $M_{j,1\oplus y_{j-\ell},0} := B_{j,0}$, $M_{j,1\oplus y_{j-\ell},1} := B_{j,1}$.

The simulator sets the correlated randomness of p and q to be $(M_{j,b,0})_{j\in[\ell],b\in\{0,1\}}$, $(M_{j,b\oplus\sigma_j,0})_{j\in\{\ell+1,\ldots,2\ell\},b\in\{0,1\}}$, $(\sigma_j)_{j\in[\ell]}$ for p and $(M_{j,b\oplus\sigma_j,1})_{j\in[\ell],b\in\{0,1\}}$, $(M_{j,b,1})_{j\in\{\ell+1,\ldots,2\ell\},b\in\{0,1\}}$, $(\sigma_j)_{j\in\{\ell+1,\ldots,2\ell\}}$ for q.

On behalf of q the simulator computes the outgoing messages to p to be $\sigma'_j := \sigma_j \oplus y_{j-\ell}$ for $j \in \{\ell+1, \ldots, 2\ell\}$, like in the protocol. On behalf of p the simulator computes the outgoing messages to q to be $(M_{j,w_j,0}, \sigma_j \oplus w_j)$ for $j \in [\ell]$ and $M_{j,\sigma'_j\oplus\sigma_j,0}$ for $j \in \{\ell+1, \ldots, 2\ell\}$, like in the protocol. Then, the simulator performs the local computations on behalf of q like in the protocol, by recovering the PSM messages $\mathbf{m}_1, \ldots, \mathbf{m}_{2\ell}$ and computing the output $\mathbf{z} := \mathsf{rec}(\mathbf{m}_1, \ldots, \mathbf{m}_{2\ell})$. Finally, the simulator updates the views of p and q accordingly.

Figure 17: Update Views for $\mathcal{F}_i$-Statement

After updating the views, Sim outputs the updated views.

Analysis. It remains to prove the following lemma.

Lemma 6.4. *The protocol $(\pi, \mathsf{Enc}, \mathsf{Rand})$ satisfies (χ, μ, ϵ)-leakage-resilient passive security with respect to the simulator* Sim.

We prove Lemma 6.4 in Section B.5. □

6.3 Instantiation: n-out-of-n Leakage-Resilient MPC with Pairwise Correlations

Let κ be a security parameter, let $f : \{0,1\}^{\ell_{\mathsf{input}}} \to \{0,1\}^{\ell_{\mathsf{out}}}$ be a circuit, and let χ be the AND-tree on n input bits.[9] Theorem 5.5 guarantees a compiler that f, χ and 1^κ outputs a tuple $(\pi', \mathsf{Enc}', \mathsf{Sim}', \mathsf{Ext}', \mathfrak{F})$, where (π', Enc') computes f in the $\mathfrak{F}$-hybrid model with $(\chi, \lfloor \log(\kappa) \rfloor, 1/\kappa)$-leakage-resilient passive security, with respect to the simulator and extractor $(\mathsf{Sim}', \mathsf{Ext}')$, and where $\mathfrak{F}$ is a family of deterministic two-party solitary-output functionalities. The running time of the generator and of $(\pi', \mathsf{Enc}', \mathsf{Sim}', \mathsf{Ext}')$ is polynomial in κ, n and $|f|$. In addition, every functionality in $\mathfrak{F}$ has depth $d = O(\log(\kappa \cdot n \cdot |f|))$ and size $s = \mathrm{polylog}(\kappa \cdot n \cdot |f|))$, and therefore, by Lemma 6.2, it has a PSM of size $s = \mathrm{poly}(\kappa, n, |f|)$.

Let Σ be the two-party $(\lfloor \log(\kappa') \rfloor, 1/\kappa')$-local leakage-resilient secret sharing scheme promised in Theorem 4.8 when applying it on security parameter $\kappa' = |\pi'| \cdot s \cdot 2^\kappa$, and observe that the running time of Σ is $\mathrm{polylog}(\kappa') = \mathrm{poly}(\kappa, n, |f|)$. Theorem 6.3 now implies a protocol $(\pi, \mathsf{Enc}, \mathsf{Rand})$ with pairwise correlations that computes f with $(\lfloor \log(\kappa) \rfloor, 1/\kappa + 2^{-\kappa})$-leakage-resilient passive security. We therefore obtain the following corollary.

[9] When n is odd, build a tree over $n+1$ inputs and set $x_{n+1} = x_1$. Note that the resulting formula is admissible for every $n > 1$.

Corollary 6.5. *For every function $f : \{0,1\}^{\ell_{\mathsf{input}}} \to \{0,1\}^{\ell_{\mathsf{out}}}$ and every (statistical) security parameter κ there is an n-out-of-n protocol $(\pi, \mathsf{Enc}, \mathsf{Rand})$ that computes f with $(\lfloor \log(\kappa) \rfloor, 1/\kappa)$-leakage resilient passive security. In addition, the running time of the protocol is $\mathrm{poly}(n, \kappa, |f|)$, and Rand generates pairwise correlations.*

7 NPSS and Leakage-Resilient NPSS

In this section we provide the formal definition of NPSS, leakage-resilient NPSS, and a construction of leakage-resilient NPSS. Throughout the section we use the notion of statistical distance, and the reader is referred to Section A.1 for basic definitions and notations.

7.1 NPSS: Definition

Before presenting the definition of NPSS, we need the following definitions for access structures and partial assignments

Definition 7.1 (Access structure)**.** *An n-party* access structure *is a monotone subset $\mathscr{A} \subseteq 2^{[n]}$. A set $S \subseteq [n]$ is called* authorized *if $S \in \mathscr{A}$, and otherwise it is called* unauthorized*.*

In some cases we will abuse notation and identify an access structure $\mathscr{A} \subseteq 2^{[n]}$ with its characteristic (monotone) function $\chi_{\mathscr{A}} : \{0,1\}^n \to \{0,1\}$. Similarly, for a monotone function $\chi : 2^{[n]} \to \{0,1\}$, we define the access structure $\mathscr{A}_\chi$ in the natural way:

$$\mathscr{A}_\chi = \{S \subseteq [n] : \chi(1_S) = 1\},$$

where 1_S is the indicator vector of the set S.

The following definition extends the NPSS definition from [AK25].

Definition 7.2 (Leakage-resilient secret sharing for an **NP** statement.)**.** *Let $\mathscr{A}$ be an n-party access structure, let κ be a (statistical) security parameter, and let μ and ϵ be functions of κ. An n-party $(\mathscr{A}, \mu, \epsilon)$-leakage-resilient secret sharing for an* **NP** *statement is a tuple of algorithms $(R, W, \mathsf{Sim}, \mathsf{Dec})$ with the following syntax:*

- *The instance mapper R is a deterministic algorithm that given a security parameter 1^κ and a circuit-SAT instance f outputs n circuit-SAT instances $(f_1, \ldots, f_n)$ over a common set of variables $\mathbf{z} = (z_1, \ldots, z_m)$.*
- *The assignment mapper W is a randomized algorithm that given a security parameter 1^κ, a circuit-SAT instance f and an assignment $\mathbf{x}$ to f, outputs a global assignment $\mathbf{y} \in \{0,1\}^m$ together with partial assignments $\mathbf{y}_1, \ldots, \mathbf{y}_n \in \{0,1,*\}^m$.*
- *The simulator Sim is a randomized algorithm that given a security parameter 1^κ, a circuit-SAT instance f, an unauthorized set $Z \notin \mathscr{A}$, and leakage functions $\tau_1, \ldots, \tau_n$, outputs partial assignments $(\mathbf{y}_i)_{i \in Z}$ and leakage $(\mathsf{leak}_i)_{i \notin Z}$.*
- *The decoder Dec is a deterministic algorithm that given a security parameter 1^κ, a circuit-SAT instance f and an authorized set S together with partial assignments $(\mathbf{y}_i)_{i \in S}$, outputs an assignment $\mathbf{x}$ for f.*

The algorithms satisfy the following properties.

- (Correctness) *If an assignment* $\mathbf{x}$ *satisfies* f *then the following holds with probability* 1*:* $W(1^\kappa, f, \mathbf{x})$ *outputs* $(\mathbf{y}, \mathbf{y}_1, \ldots, \mathbf{y}_n)$ *such that each partial assignment* $\mathbf{y}_i$ *is consistent with* $\mathbf{y}$ *on all Boolean values, and* $\mathbf{y}_i$ *satisfies* f_i*, where* $(f_1, \ldots, f_n) = R(1^\kappa, f)$*.*

- (Privacy) *If* $\mathbf{x}$ *satisfies* f *then for every* κ*, every unauthorized set* $Z \notin \mathscr{A}$*, and every leakage functions* $\tau_1, \ldots, \tau_n$ *where* τ_i *takes a partial assignment* $\mathbf{y}_i$ *and outputs* $\mu(\kappa)$ *bits, it holds that the random variables*
$$((\mathbf{y}_i)_{i\in Z}, (\tau_i(\mathbf{y}_i))_{i\notin Z}) \quad \text{and} \quad \left((\mathbf{y}'_i)_{i\in Z}, (\tau_i(\mathbf{y}'_i))_{i\notin Z}\right)$$
are $\epsilon(\kappa)$*-close in statistical distance, where* $(\mathbf{y}, \mathbf{y}_1, \ldots, \mathbf{y}_n) \leftarrow W(1^\kappa, f, \mathbf{x})$*, and* $(\mathbf{y}'_1, \ldots, \mathbf{y}'_n) \leftarrow \mathsf{Sim}(1^\kappa, f, Z)$*.*

- (Recovery) *For any authorized set* $S \in \mathscr{A}$*, and for any partial assignments* $(\mathbf{y}_i)_{i\in S}$ *that are pairwise consistent on Boolean values, it holds that if every* $\mathbf{y}_i$ *satisfies* f_i *then* $\mathsf{Dec}(1^\kappa, f, S, (\mathbf{y}_i)_{i\in S})$ *outputs an assignment* $\mathbf{x}$ *that satisfies* f*.*

The special case where $\mu = 0$ *(i.e., there is no leakage) and* $\epsilon = 0$ *(i.e., the simulation is perfect) corresponds to the notion of (non-leakage resilient) NPSS that as defined in [AK25].*

We say that an n-party NPSS is a *k-out-of-n NPSS* if the access structure $\mathscr{A} \subseteq 2^{[n]}$ contains all subsets of $[n]$ of size at least k, and no subset of $[n]$ of size less than k.

7.2 From Leakage-Resilient MPC to Leakage-Resilient NPSS

Recall that a monotone formula χ is *admissible* if (1) it uses only AND and OR gates with fan-in 2, (2) it uses no constants, (3) all its input gates are AND gate, and no input gate takes the same input variable twice. (See Definition 5.4.)

Theorem 7.3. *There is a deterministic compiler that takes as an input a depth-d admissible monotone formula χ with n input bits and a single output bit, and outputs the description of an n-party $(\mathscr{A}_\chi, \mu(\kappa) := \lfloor \log(\kappa) \rfloor, \epsilon(\kappa) := 1/\kappa)$-leakage-resilient secret sharing for an* **NP** *statement $(R, W, \mathsf{Sim}, \mathsf{Dec})$. The compiler runs in time $\mathrm{poly}(n \cdot 2^d)$. The running time of the algorithms $(R, W, \mathsf{Sim}, \mathsf{Dec})$ is $\mathrm{poly}(\kappa \cdot 2^d \cdot n \cdot |f|)$, where κ is the (statistical) security parameter, f is the circuit-SAT instance that the algorithms take as an input, and $|f|$ is the circuit-size of f measured by the number of wires.*

Proof. We begin by presenting some notation.

Notation. Fix any formula χ, and some security parameter κ. We let C be the MPC-compiler promised by Theorem 5.5, and we denote by $C_{\chi,\kappa}$ the compiler C when the input-formula (acsess structure) is fixed to be χ, so $C_{\chi,\kappa}$ only takes as an input a function f.

We assume without loss of generality that every protocol π generated by $C_{\chi,\kappa}$ satisfies the following syntactic property. Before any call to a two-party solitary-output ideal functionality $\mathcal{F}$, the sender locally computes all the possible outcomes of $\mathcal{F}$. Formally, if the sender S holds the input $\mathbf{a}$, and the receiver R holds the input $\mathbf{b}$, the sender S first locally computes the values $\mathbf{c}_{\mathbf{b}'} := \mathcal{F}(\mathbf{a}, \mathbf{b}')$ for every possible input $\mathbf{b}'$ of the receiver, and only then executes the $\mathcal{F}$ instance with R. We call $(\mathbf{c}_{\mathbf{b}'})_{\mathbf{b}'}$ the *possible outputs* of the $\mathcal{F}$ instance, and we call $\mathbf{c}_{\mathbf{b}}$ the *chosen output* of

the $\mathcal{F}$ instance. We emphasize that $(\mathbf{c}_{\mathbf{b}'})_{\mathbf{b}'}$ are variables that are owned by the sender S. Recall that for every circuit f, every two-party solitary-output functionality $\mathcal{F}$ in the generated protocol π has input-length of $O(\log(\kappa \cdot n \cdot 2^d \cdot |f|))$, and therefore the number of possible outputs is at most $\mathrm{poly}(\kappa, n, 2^d, |f|)$. In addition, for every $\mathsf{transmit}(p_i, p_j, x_1, x_2)$ we call x_1 the *transmitted message* and x_2 the *received message*.

Given a vector of values $\mathbf{y}$ for all the variables of the protocol, the *extended view* of a party p_i consists of all the variables that belong to p_i and, in addition, all the "transmitted messages" that are directed to p_i and all the "chosen output" of each $\mathcal{F}$ call in which p_i plays the receiver. (Note that the transmitted messages and the chosen outputs are not owned by p_i.)

Construction overview. Given a circuit-SAT instance f, we use $C_{\chi,\kappa}$ to generate a protocol $\Pi' = (\pi', \mathsf{Enc}')$ that computes f with $(\chi, \lfloor \log(\kappa) \rfloor, 1/\kappa)$-leakage-resilient dual security, together with the corresponding simulator Sim' and the extractor Ext'. The global assignment $\mathbf{y}$ consists of all the variables $\mathcal{X}$ in an honest execution of Π'. The i-th circuit f_i takes as an input a partial assignment $\mathbf{y}_i$ that contains the extended view of the i-th party p_i and verifies that it is a valid accepting transcript. That is, f_i checks that the messages transmitted to the i-th client are consistent with the received messages; that for every $\mathcal{F}$-statement where the outputs are directed to p_i, the chosen output is consistent with the output of $\mathcal{F}$; and that the local computations performed by p_i are consistent with π' and that the final output is 1. To sample a global assignment $\mathbf{y}$, we invoke the protocol Π' on $\mathbf{x}$. We continue with a formal description of the construction.

The construction. The algorithms $(R, W, \mathsf{Sim}, \mathsf{Dec})$ are defined as follows:

- **Instance mapper.** The algorithm R takes as an input a security parameter 1^κ and an instance of circuit-SAT f. The algorithm executes $C_{\chi,\kappa}(f)$ to obtain $(\pi', \mathsf{Enc}', \mathsf{Sim}', \mathsf{Ext}', \mathfrak{F})$ over the variable space $\mathcal{X}$. It outputs n circuits $f_1, \ldots, f_n$, where f_i is defined as follows.
  - *(Inputs)* It takes as an input a partial assignment $\mathbf{y}_i$ that is parsed as the values of the variables in $\mathcal{X}$. We denote by $\mathbf{y}_i[x] \in \{0, 1, *\}$ the value of the variable $x \in \mathcal{X}$ according to $\mathbf{y}_i$, and for a subset of variables $X \subseteq \mathcal{X}$ we denote $\mathbf{y}_i[X] = (\mathbf{y}_i[x])_{x \in X}$.
  - *(Computation)* For two bits $a, b \in \{0, 1\}$, let $\mathsf{eq}(a, b)$ be a sub-circuit that returns 1 if $a = b$, and 0 otherwise, i.e., $\mathsf{eq}(a, b) = (a \wedge b) \vee (\neg a \wedge \neg b)$. Observe that $\mathsf{eq}(a, *) = \mathsf{eq}(*, a) = *$ for every $a \in \{0, 1, *\}$.

    Furthermore, for two strings $\mathbf{a}, \mathbf{b} \in \{0, 1\}^\ell$, let $\mathsf{vEQ}(\mathbf{a}, \mathbf{b}) = \wedge_{i \in [\ell]} \mathsf{eq}(a_i, b_i)$. We observe that if $\mathbf{a} \in \{0, 1, *\}^\ell \setminus \{0, 1\}^\ell$ (i.e., if $a_i = *$ for some $i \in [\ell]$) then for every $\mathbf{b} \in \{0, 1, *\}^\ell$ it holds that $\mathsf{vEQ}(\mathbf{a}, \mathbf{b}) = \mathsf{vEQ}(\mathbf{b}, \mathbf{a}) \in \{0, *\}$.

    The circuit verifies that $\mathbf{y}_i$ corresponds to a valid extended view of p_i, by performing the following local computations:
    1. For every variables x in the view of p_i, it verifies that $\mathbf{y}_i[x] \neq *$ by computing $\mathsf{eq}(\mathbf{y}_i[x], 0) \vee \mathsf{eq}(\mathbf{y}_i[x], 1)$.
    2. For every $\mathsf{comp}(p_i, \mathsf{op}, (x_1, x_2), x)$ statement with $\mathsf{op} \in \{+, \times\}$, it verifies that the statement was computed correctly in the view, by computing $\mathsf{eq}(\mathsf{op}(\mathbf{y}_i[x_1], \mathbf{y}_i[x_2]), \mathbf{y}_i[x])$.
    3. For every abort statement $\mathsf{abort}(p_i, x)$ it verifies that c_i did not raise the abort flag, i.e., it computes $\mathsf{eq}(\mathbf{y}_i[x], 0)$.

4. For every $\mathsf{transmit}(c_j, c_i, x_1, x_2)$ it verifies that the transmitted message is equal to the received message by computing $\mathsf{eq}(\mathbf{y}_i[x_1], \mathbf{y}_i[x_2])$.
5. Consider a statement $\mathsf{func}(\mathcal{F}, p_j, p_i, X_1, X_2, X_3)$ where $\mathcal{F} \in \mathfrak{F}$ is a two-party solitary-output functionality, and where the sender p_j inputs the values of the variables in X_1, the receiver p_i inputs the values of the variables in X_2 and stores the output of $\mathcal{F}$ in the variables of X_3. The circuit f_i verifies that the actual output (i.e., the value of X_3) is equal to the value of the chosen output of the $\mathcal{F}$ instance. That is, we denote by $(\mathbf{c_b})_\mathbf{b}$ the possible outputs of $\mathcal{F}$ and by ℓ_{input} the length of Bob's input to the functionality, and the circuit performs the following computation

$$\bigvee_{\mathbf{b}\in\{0,1\}^{\ell_{\mathsf{input}}}} (\mathsf{vEQ}(\mathbf{b}, \mathbf{y}_i[X_2]) \wedge \mathsf{vEQ}(\mathbf{y}_i[X_3], \mathbf{y}_i[X(\mathbf{c_b})]) \,,$$

where $X(\mathbf{c_b})$ are the variables that correspond to the possible output $\mathbf{c_b}$. We recall that the receiver's input to the functionality has logarithmic length, $\ell = O(\log(\kappa \cdot n \cdot 2^d \cdot |f|))$, and therefore the sub-circuit performing this local computation has size polynomial in $(\kappa \cdot n \cdot 2^d \cdot |f|)$.
6. The protocol π' contains a single output instruction of the form $\mathsf{out}(p_i, 1, x)$. The circuit verifies that $\mathbf{y}_i[x] = 1$, i.e., it computes $\mathsf{eq}(1, \mathbf{y}_i[x])$.

– *(Output)* The output is the AND over all the local computations.

- **Assignment mapper.** The algorithm W takes as an input a security parameter 1^κ, a circuit-SAT instance f, and an assignment $\mathbf{x}$ to f. The algorithm executes $C_{\chi,\kappa}(f)$ to obtain $(\pi', \mathsf{Enc}', \mathsf{Sim}', \mathsf{Ext}', \mathfrak{F})$ over the variable space $\mathcal{X}$. It then executes the protocol $\Pi' = (\pi', \mathsf{Enc}')$ on $\mathbf{x}$. The global assignment $\mathbf{y}$ is set to be the values of all the variables in $\mathcal{X}$. The i-th partial assignment $\mathbf{y}_i$ is equal to $\mathbf{y}$ for every variable x in the extended view of p_i, and in all other variables it is equal to $*$. The algorithm outputs $(\mathbf{y}, \mathbf{y}_1, \ldots, \mathbf{y}_n)$.

- **Simulator.** The algorithm Sim as an input a security parameter 1^κ, a circuit-SAT instance f, an unauthorized set $Z \notin \mathscr{A}_\chi$ and leakage functions $\tau_1, \ldots, \tau_n$. The algorithm executes $C_{\chi,\kappa}(f)$ to obtain $(\pi', \mathsf{Enc}', \mathsf{Sim}', \mathsf{Ext}', \mathfrak{F})$ over the variable space $\mathcal{X}$. It executes $(\mathsf{view}_i)_{i\in[n]} \leftarrow \mathsf{Sim}'(1^\kappa, Z, 1)$. The algorithm Sim extends view_i into the extended view view'_i by (1) setting the transmitted message to be equal to the received message for every $\mathsf{transmit}(p_j, p_i, x_1, x_2)$ statement, and (2) for every $\mathcal{F}$-statement $\mathsf{func}(\mathcal{F}, p_j, p_i, X_1, X_2, X_3)$ in which p_i acts as the receiver with input $\mathbf{b}$ and output $\mathbf{c}^*$, setting the chosen output $\mathbf{c_b}$ to be $\mathbf{c}^*$. The extended views $(\mathsf{view}'_i)_{i\in[n]}$ are then used to create the partial assignments $(\mathbf{y}_i)_{i\in[n]}$, and outputs $(\mathbf{y}_i)_{i\in[n]}$.

- **Decoder.** The algorithm Dec takes as an input a security parameter 1^κ, circuit-SAT instance f, and an authorized set $S \in \mathscr{A}_F$ together with partial assignments $(\mathbf{y}_i)_{i\in S}$. The algorithm executes $C_{\chi,\kappa}(f)$ to obtain $(\pi', \mathsf{Enc}', \mathsf{Sim}', \mathsf{Ext}', \mathfrak{F})$ over the variable space $\mathcal{X}$. For every $i \in S$ the decoder parses $\mathbf{y}_i$ as the extended view of p_i, and sets view_i to be the view of p_i. The decoder computes $\mathbf{x} = \mathsf{Ext}'(S, (\mathsf{view}_i)_{i\in S})$ and outputs $\mathbf{x}$.

We continue with an analysis. Fix any (statistical) security parameter κ, circuit-SAT instance f, let $R(1^\kappa, f) = (f_1, \ldots, f_n)$, and let $(\pi', \mathsf{Enc}', \mathsf{Sim}', \mathsf{Ext}', \mathfrak{F})$ be the corresponding protocol, input encoder, simulator and extractor that are generated by executing $C_{\chi,\kappa}(f)$. We denote $\Pi' = (\pi', \mathsf{Enc}')$.

Correctness. Let $\mathbf{x}$ be a satisfying assignment of f. The global assignment $\mathbf{y}$ generated by $W(1^\kappa, f, \mathbf{x})$ corresponds to an honest execution of $\Pi' = (\pi', \mathsf{Enc}')$ in which the outputs of all the clients is 1, and every $\mathbf{y}_i$ corresponds to the extended view of the i-th client. Therefore $\mathbf{y}_i$ satisfies f_i. This concludes the correctness analysis.

Privacy. Let $\mathbf{x}$ be a satisfying assignment of f and fix any unauthorized set $Z \notin \mathscr{A}_\chi$, i.e., $\chi(Z) = 0$. Fix any leakage functions $\tau_1, \ldots, \tau_n$ that output μ bits, and define the leakage function ν_i in the following way: ν_i takes the view view_i of the i-th party in an execution of (π', Enc'), computes the corresponding extended view, and then the corresponding partial assignment $\mathbf{y}_i$ and outputs $\tau_i(\mathbf{y}_i)$. By the privacy of Π' it holds that

$$(\mathsf{view}_i)_{i\in Z}, (\nu_i(\mathsf{view}_i))_{i\notin Z} \equiv_{1/\kappa} (\mathsf{view}'_i)_{i\in Z}, (\nu_i(\mathsf{view}'_i))_{i\notin Z},$$

where $(\mathsf{view}_1, \ldots, \mathsf{view}_n)$ are generated in an execution of Π' on input $\mathbf{x}$, and $(\mathsf{view}'_1, \ldots, \mathsf{view}'_1) \leftarrow \mathsf{Sim}'(Z, 1)$. Observe that $\nu_i(\mathsf{view}_i) = \tau_i(\mathbf{y}_i)$ where $\mathbf{y}_i$ is the partial assignment corresponding to the extended view of view_i. Consider the procedure that takes $(\mathsf{view}_i)_{i\in Z}, (\nu_i(\mathsf{view}_i))_{i\notin Z}$, for every $i \in Z$ computes the extended view of view_i and the corresponding partial assignment $\mathbf{y}_i$ and outputs $(\mathbf{y}_i)_{i\in Z}, (\nu_i(\mathsf{view}_i))_{i\notin Z}$. By applying this procedure on both sides we obtain

$$(\mathbf{y}_i)_{i\in Z}, (\tau_i(\mathbf{y}_i))_{i\notin Z} \equiv_{1/\kappa} (\mathbf{y}'_i)_{i\in Z}, (\nu_i(\mathbf{y}'_i))_{i\notin Z}.$$

Finally, we observe that the LHS random variables $(\mathbf{y}_1, \ldots, \mathbf{y}_n)$ have the same distribution as $W(1^\kappa, f, \mathbf{x})$, and the RHS random $(\mathbf{y}'_1, \ldots, \mathbf{y}'_n)$ have the same distribution as $\mathsf{Sim}(1^\kappa, f, Z)$. This concludes the proof of privacy.

Recovery. Fix any authorized set $S \in \mathscr{A}_\chi$, and any partial assignments $(\mathbf{y}_i)_{i\in S}$ that are pairwise consistent on Boolean values, and assume that $\mathbf{y}_i$ satisfies $\hat{f}_i$ for all $i \in S$. This implies that (1) for every variable x in the view of p_i it holds that $\mathbf{y}_i[x] \in \{0, 1\}$, (2) the view defined by $\mathbf{y}_i$ corresponds to an honest execution of π' by p_i, (3) the private channels and the $\mathcal{F}$-statements worked correctly between every pair of parties in S, and (4) the outputs of all the parties in S is 1, and no party raised the abort flag. Let $(\mathsf{view}_i)_{i\in S}$ correspond to the views of the parties in S according to $(\mathbf{y}_i)_{i\in S}$.

We note that there exists an active adversarial behavior $\tilde{\pi}'$ that can result in $(\mathsf{view}_i)_{i\in S}$. Indeed, for every statement $\mathsf{transmit}(p_i, p_j, x_1, x_2)$ where $i \notin S$ and $j \in S$, mark the statement by the value $\mathbf{y}_j[x_2]$, and for every statement $\mathsf{func}(\mathcal{F}, p_i, p_j, X^1, X^2, X^3)$ where $i \notin S$ and $j \in S$, mark the statement by the value $\mathbf{y}_j[X^3]$. Now, if the parties in S execute $\tilde{\pi}'$ with the same randomness as in $(\mathsf{view}_i)_{i\in S}$ then they obtain the same views. In addition, since $S \in \mathscr{A}_\chi$ then $\chi(S) = 1$, and let $Z = [n] \setminus S$. This implies that $\mathsf{Dec}(1^\kappa, f, S, (\mathsf{view}_i)_{i\in S}) = \mathsf{Ext}'(S, (\mathsf{view}_i)_{i\in S})$ outputs $\mathbf{x}$ such that $f(\mathbf{x}) = 1$. This concludes the recovery analysis and the proof of the theorem. □

7.3 Constructing t-out-of-n Leakage-Resilient NPSS

For $1 \leq t \leq n$, let Th^n_t be the threshold formula that takes n input bits, outputs 1 if the Hamming weight of the input is at least t, and outputs 0 otherwise. To construct t-out-of-n leakage-resilient NPSS, it only remains to show that Th^n_t applies a logarithmic-depth admissible formula.

First, we note that an explicit construction of a logarithmic-depth formula with AND and OR gates with fan-in 2 and with no constants, follows from the classical constructions of logarithmic-depth sorting networks [AKS83, Pat90, Sei09].[10] However, these constructions do not necessarily satisfy the condition that all the input gates are AND gates. The following claim shows that these constructions can be modified to obtain an explicit construction of a logarithmic-depth admissible formula.

Claim 7.4. *For every integers $2 \leq t \leq n$ there is an explicit construction of a logarithmic-depth admissible formula g for Th_t^n.*

Proof. For $t = n$ we can take g to be an AND tree if n is even. If n is odd, we can build a tree over $n+1$ inputs and set $x_{n+1} = x_1$. It is not hard to verify that the resulting formula is admissible. From now on, we assume that $2 \leq t \leq n-1$. As we've mentioned, the classical works of [AKS83, Pat90, Sei09] imply that for every $1 \leq T \leq N$ there exists an explicit construction of a logarithmic-depth formula for Th_T^N that uses only AND and OR gates with fan-in 2, and uses no constants (however, some of its input gates might be OR gates).

Set $N = \binom{n}{3}$. First, we consider the following sub-formula H with Th_2^3 gates, that takes n input bits $x_1, \ldots, x_n$ and outputs N output bits $(y_{\{i,j,k\}})_{1 \leq i < j < k \leq n}$, where $y_{\{i,j,k\}} = \mathsf{Th}_2^3(x_i, x_j, x_k)$, i.e., it is equal to the majority of the three bits. By symmetry, the Hamming weight of the output depends only on the Hamming weight of the input, and let us denote by $w(\ell)$ the Hamming weight of the output when the Hamming weight of the input is ℓ. It is not hard to see that the function w is strictly increasing in the interval $2 \leq t \leq n-1$, i.e., $w(t-1) < w(t)$ for every $2 \leq t \leq n-1$. (Indeed, when we flip a zero to one in an assignment of weight $t-1$ for $1 \leq t-1 \leq n-2$ at least one zero output of H is increased to one, and all one outputs remain unchanged). Therefore, $\mathsf{Th}_t^n = \mathsf{Th}_{w(t)}^N \circ H$, and we obtained a formula for Th_t^n where all the input gates are Th_2^3 gates, the remaining gates are AND and OR gates with fan-in 2, and the formula uses no constants. Finally we observe that $\mathsf{Th}_2^3(x_1, x_2, x_3)$ can be written as $(x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3)$, and therefore we can replace each Th_2^3 gate with constant-size admissible formula. Overall, the resulting formula is also admissible, as required. □

The following corollary follows.

Corollary 7.5. *There is a deterministic compiler C_{Th} that takes as an input two integers $2 \leq t \leq n$, runs in time $\mathrm{poly}(n)$ and outputs the description of a t-out-of-n $(\mu(\kappa) = \lfloor \log(\kappa) \rfloor, \epsilon(\kappa) = 1/\kappa)$-NPSS $(R, W, \mathsf{Sim}, \mathsf{Dec})$. The running time of all algorithms in $(R, W, \mathsf{Sim}, \mathsf{Dec})$ is $\mathrm{poly}(\kappa \cdot n \cdot |f|)$ where f is the circuit-SAT instance that the algorithms take as an input, and $|f|$ is the circuit-size of f measured by the number of wires.*

8 Application: NIZK Amplification

In this section we prove our results regarding NIZK amplification. We begin with some useful background (Section 8.1) including notation, tools and simple NIZK transformations that allow us

[10] At a high level, sorting networks are Boolean circuits with *comparators* as gates, where each comparator takes two bits and outputs them in sorted order, i.e., $\mathsf{comp}(x, y) = (\min(x, y), \max(x, y))$. Since every comparator can be implemented by a constant-size monotone circuit that uses no constants $\mathsf{comp}(x, y) = ((x \wedge y), (x \vee y))$, each output bit of the sorting network can be described as a logarithmic-depth monotone circuit that uses no constants. We can now take Th_k^n to be the $(n-k+1)$-output bit of the network.

to freely move between zero-knowledge and witness-indistinguishability (WI). Next, we present and analyze our main amplifier (Section 8.2). This amplifier reduces the soundness (or proof-of-knowledge) error to negligible, and the WI error to an arbitrary inverse polynomial. We then use another amplifier (Section 8.3) to further reduces the WI error to negligible. Finally, we explain how to combine all the ingredients to a single amplifier that transforms a weak NIZK with inverse-polynomial gap into a NIZK (Section 8.4).

We make extensive use of computational indistinguishability, and use the notation $A \approx B$ to indicate that the ensembles A and B are computationally indistinguishability. As already mentioned, the notations $A \equiv_\epsilon B$ and $A \equiv B$ indicate that the ensemble are ϵ-close in statistical distance, and identically distributed, respectively. For more background, see Section A.

8.1 Useful Primitives and Lemmas

8.1.1 Non-Interactive Commitments and Trapdoor Commitments

Definition 8.1 (Non-interactive commitment)**.** *A non-interactive commitment (NICOM) schem is a tuple of polynomial-time algorithms* $(\mathsf{Gen}, \mathsf{Commit}, \mathsf{open})$ *with the following syntax:*

- $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\kappa)$. *The randomized set-up algorithm takes as an input a security parameter* κ *and outputs public parameters* pp*.*
- $(C, o) \leftarrow \mathsf{Commit}_{\mathsf{pp}}(1^\kappa, m)$. *The randomized (honest) commitment generating algorithm takes as input the public parameters* pp*, the security parameter* 1^κ *and a bit* $m \in \{0,1\}$*. It outputs a commitment* C *to* m*, along with an opening* o*.*
- $b = \mathsf{VOpen}_{\mathsf{pp}}(1^\kappa, C, o, m)$. *The deterministic verify opening algorithm takes as input the public parameters* pp*, the security parameter* 1^κ*, a commitment* C*, opening* o *and a message* $m \in \{0,1\}$*. It outputs a bit* b *indicating accept or reject.*

We require the following properties:

- **Correctness:** *For every* $\kappa, \mathsf{pp}, x, r$*, it holds that* $\mathsf{VOpen}_{\mathsf{pp}}(1^\kappa, \mathsf{Commit}_{\mathsf{pp}}(1^\kappa, x; r)) = x$*.*
- **Binding:** *For every family of polynomial-size non-uniform adversaries* $\mathcal{A} = \{\mathcal{A}_\kappa\}$ *and every security parameter* κ*, with probability at most* $\epsilon = \mathsf{negl}(\kappa)$ *over a uniform choice of* pp*, the tuple* $(C, o, o') := \mathcal{A}_\kappa(\mathsf{pp})$ *satisfies* $\mathsf{VOpen}_{\mathsf{pp}}(1^\kappa, C, o, 0) = \mathsf{VOpen}_{\mathsf{pp}}(1^\kappa, C, o', 1) = 1$*.*

  The scheme is statistically binding*, if the above holds even for inefficient adversaries, and* perfectly binding *if, in addition,* $\epsilon = 0$*.*
- **Hiding:** *For every family of non-uniform adversaries* $\mathcal{A} = \{\mathcal{A}_\kappa\}$*, every security parameter* κ*, every* pp*, the distinguishing gap*

$$\left| \Pr_{(C,o) \leftarrow \mathsf{Commit}_{\mathsf{pp}}(1^\kappa, 0)} [\mathcal{A}_\kappa(\mathsf{pp}, C) = 1] - \Pr_{(C,o) \leftarrow \mathsf{Commit}_{\mathsf{pp}}(1^\kappa, 1)} [\mathcal{A}_\kappa(\mathsf{pp}, C) = 1] \right| \leq \epsilon(\kappa)$$

  for some negligible $\epsilon(\cdot)$*. The scheme is* statistically hiding *if the above holds even for inefficient adversaries.*

For ease of reading, we typically omit the security parameter and the public parameters from the algorithms. By default, the security parameter is set according to the global security parameter that is being used by the system.

Instantiations. We consider the following instantiations of NICOM from symmetric-key primitives. The first construction that we consider, due to Naor [Nao91], is based on one-way functions and requires a public parameter pp in the form of a uniformly random string. The construction provides statistically-binding computationally-hiding commitments. The second construction that we consider, due to [DPP98, HM96] is based on collision resistant hash function. It requires the sampling of a collision resistant hash function as a public parameter pp, and it is computationally binding and statistically hiding.

Trapdoor commitments. For the construction of the amplifier we also need the following notion of *trapdoor commitments*, due to [FS89].

Definition 8.2. *A* trapdoor commitment *scheme is a tuple of polynomial-time algorithms* $(\mathsf{Gen}, \mathsf{Gen}', \mathsf{Commit}, \mathsf{VOpen}, \mathsf{TCom})$ *with the following syntax:*

- $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\kappa)$. *The randomized (honest) set-up algorithm takes as input a security parameter* $\kappa \in \mathbb{N}$ *and outputs public parameters* pp.
- $(\mathsf{pp}, \tau) \leftarrow \mathsf{Gen}'(1^\kappa)$. *The randomized ("fake") set-up algorithm takes as input a security parameter* $\kappa \in \mathbb{N}$ *and outputs public parameters* pp *and a trapdoor* τ.
- $(C, o) \leftarrow \mathsf{Commit}_{\mathsf{pp}}(m)$. *The randomized (honest) commitment generating algorithm takes as input a bit* $m \in \{0,1\}$. *It outputs a commitment* C *to* m, *along with an opening* o. *We denote by* $\ell(\kappa)$ *the length of* C *and* o.
- $b = \mathsf{VOpen}^z_{\mathsf{pp}}(C, o, m)$. *The deterministic verify opening algorithm takes as input a commitment* C, *opening* o *and a message* $m \in \{0,1\}$. *It outputs a bit* b *indicating accept or reject.*
- $(C, o_0, o_1) \leftarrow \mathsf{TCom}_{\mathsf{pp}}(\tau)$. *The randomized ("fake") commitment generation algorithm takes as input the trapdoor* τ. *It outputs a commitment* $C \in \{0,1\}^{\ell(\kappa)}$, *together with a pair of openings* (o_0, o_1).

We require the following properties:

1. *The honest algorithms* $(\mathsf{Gen}, \mathsf{Commit}, \mathsf{VOpen})$ *form a statistically-binding non-interactive commitment scheme. We refer to this scheme as the* associated honest NICOM.
2. *The real and fake mode are indistinguishable. Formally, for every* $n = \mathrm{poly}(\kappa)$ *and* $(b_1, \ldots, b_n) \in \{0,1\}^n$ *it holds that*
$$(\mathsf{pp}, (C_i, o_i)) \approx (\mathsf{pp}', (C'_i, o'_i)),$$
where $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\kappa)$, $(C_i, o_i) \leftarrow \mathsf{Commit}_{\mathsf{pp}}(b_i)$, $(\mathsf{pp}', \tau') \leftarrow \mathsf{Gen}'(1^\kappa)$, $(C_i, o_{i,0}, o_{i,1}) \leftarrow \mathsf{TCom}_{\mathsf{pp}}(\tau)$ *and* $o'_i := o_{i,b_i}$.

A construction of trapdoor commitments is described in [FS89]. As observed by [BG24], in the CRS model, where non-interactive commitments follow from one-way functions [Nao91], we can obtain trapdoor commitments from one way functions.

Theorem 8.3. *Assuming the existence of one-way functions, there exists a trapdoor commitment scheme.*

8.1.2 XOR Lemmas and Guessing Games

Notation. Let A and B be two algorithms, and let $b \in \{0,1\}$ be a bit. We denote by $F := \langle A, B \rangle_b$ the algorithm that is equal to A if $b = 0$, and is equal to B if $b = 1$.

The guessing game. Let A and B be a pair of randomized algorithms with the same input domain. In the n-wise guessing game $\mathsf{Game}(A, B, n)$ a guesser $\mathcal{G}$ has oracle access to n algorithms $F_1, \ldots, F_n$ that are chosen in the following way: pick $\mathbf{s} = (s_1, \ldots, s_n) \leftarrow \{0,1\}^n$ and then set $F_1 := \langle A, B \rangle_{s_1}, \ldots, F_n := \langle A, B \rangle_{s_n}$. The guesser $\mathcal{G}$ performs a single call $\mathbf{x}_i$ to each oracle F_i and based on the answers from the oracle, the guesser outputs a bit, which is the guesser's prediction to the bit $\bigoplus_{i \in [n]} s_i$.[11] For our purpose, it suffices to consider only guessers with non-adaptive queries to the oracles.

We define the advantage of $\mathcal{G}$ to be $\mathsf{Guess}^{\mathcal{G}}(A, B, n) = 2 \Pr[\mathcal{G}^{F_1, \ldots, F_n} = \bigoplus_{i \in [n]} s_i] - 1$. We denote by $\mathsf{Guess}(A, B, n)$ the best advantage $\mathsf{Guess}^{\mathcal{G}}(A, B, n)$ taken over all (computationally-unbounded) guessers $\mathcal{G}$. For an integer s we denote by $\mathsf{Guess}^s(A, B, n)$ the best advantage $\mathsf{Guess}^{\mathcal{G}}(A, B, n)$ taken over all guessers $\mathcal{G}$ of circuit size s. Observe that $\mathsf{Guess}(A, B, n) \geq 0$ and $\mathsf{Guess}^s(A, B, n) \geq 0$ since we can always flip the output bit to get an adversary with positive guessing advantage.

Information-theoretic variant. First we present the information-theoretic variant of the XOR lemma, which is a special case of [Tes10, Theorem 3.13].

Theorem 8.4. *Let A and B be randomized algorithms and let n be a positive integer. Then $\mathsf{Guess}(A, B, n) \leq \mathsf{Guess}(A, B, 1)^n$.*

Computational variant. We continue with the computational variant of the XOR lemma.

Theorem 8.5. *Let $\ell_{\mathsf{input}}, \ell_{\mathsf{out}} : \mathbb{N} \to \mathbb{N}$ be functions, and let $\{A_\kappa\}_{\kappa \in \mathbb{N}}$ and $\{B_\kappa\}_{\kappa \in \mathbb{N}}$ be families of randomized circuits with input length $\ell_{\mathsf{input}}(\kappa)$ and output length $\ell_{\mathsf{out}}(\kappa)$. Let $\delta(\kappa) : \mathbb{N} \to [0,1]$ be a function that is bounded-away from 1 (i.e., $\delta(\kappa) < 1 - \frac{1}{p(\kappa)}$ for some polynomial p and all sufficiently large κ's). Let $s(\kappa) : \mathbb{N} \to \mathbb{N}$ be a size function, and assume that for all κ it holds that*

$$\mathsf{Guess}^{s(\kappa)}(A_\kappa, B_\kappa, 1) \leq \delta(\kappa).$$

Then, for every function $n(\kappa) : \mathbb{N} \to \mathbb{N}$, every function $\epsilon : \mathbb{N} \to [0,1]$, and all κ it holds that

$$\mathsf{Guess}^{s'(\kappa)}(A_\kappa, B_\kappa, n(\kappa)) \leq \delta'(\kappa),$$

where $\delta'(\kappa) = e^{n(\kappa)-1} \cdot (\delta(\kappa)^{n(\kappa)} + \epsilon(\kappa))$ and

$$s'(\kappa) = \frac{s(\kappa)}{\mathrm{poly}(\ell_{\mathsf{out}}(\kappa) \cdot n(\kappa)/\epsilon(\kappa))} - \mathrm{poly}(\ell_{\mathsf{out}}(\kappa) \cdot \ell_{\mathsf{input}}(\kappa) \cdot n(\kappa)).$$

The proof of Theorem 8.5 relies on Yao's XOR lemma. Indeed, the simpler case where the randomized algorithms A and B take no input (and so is F) follows immediately from the original XOR lemma. Some care is needed in order to deal with algorithms that take input. Specifically, we exploit that fact that the adversary selects its inputs $(q_1, \ldots, q_n)$ to $F_1, \ldots, F_n$ in a non-adaptive way

[11] Formally, we model $\mathcal{G}$ as a circuit with oracle gates to $F_1, \ldots, F_n$.

to reduce the problem to an inputless setting. Roughly speaking, we let each F sample one query q_i out of $(q_1, \ldots, q_n)$ and return the corresponding answer (together with q_i). If we are "lucky" and each q_i is selected exactly once among the n calls to F, we recover the "right" experiment (as we can reorder the calls). Since this good event happens with probability of e^{-n}, the corresponding bound δ' loses this additional multiplicative factor compared to the original statement. We prove Theorem 8.5 in Section C.1.

Previous works have considered the case where A and B take inputs, but assumed in addition that the original (single-instance) guessing game is secure against adversaries making multiple queries to F. Indeed, in this case one can avoid the extra loss, assuming that the adversary is allowed to ask $\Omega(1/\epsilon(\kappa)^2)$ queries in the single instance game. (See, e.g., [Tes10, Theorem 3.14 and Theorem 3.15]). These variants cannot be used when the original guessing game is insecure against polynomially many queries, like in our case.

8.1.3 Indistinguishability Lemma

Notation. For an NIP $\Pi = (\mathsf{Gen}, \mathcal{P}, \mathcal{V})$, a circuit f and an assignment $\mathbf{x}$, we denote by $\mathsf{Prove}_\kappa(f, \mathbf{x})$ the random variable (crs, π) where $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\kappa)$ and $\pi \leftarrow \mathcal{P}(1^\kappa, \mathsf{crs}, f, \mathbf{x})$. We think of Prove as a randomized algorithm, so that $\mathsf{Prove}_\kappa(f, \mathbf{x})$ uses a random string sampled from a distribution $\mathcal{R}^p_\kappa(f, \mathbf{x})$. If Π satisfies zero-knowledge or WI with respect to a simulator Sim, we denote by $\mathcal{R}^s_\kappa(f)$ the domain of randomness from which $\mathsf{Sim}(1^\kappa, f)$ samples randomness.

Statistical variant. We begin with the following well known fact (see, e.g., [MT10]).

Fact 8.6. *Let X and Y be random variables that satisfy $X \equiv_\delta Y$. Then there exists a pair of events A and B with probability $1 - \delta$ each, such that $X|A \equiv Y|B$.*

The following claim follows immediately from the fact.

Claim 8.7. *Let $\Pi = (\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ be an NIP that satisfies $\epsilon_w(\kappa)$-statistical-WI and let Sim be the corresponding (inefficient) simulator. For every polynomial p there exists $\kappa_0 \in \mathbb{N}$ such that for every $\kappa \geq \kappa_0$, and every circuit f and assignment $\mathbf{x}$ that satisfy $f(\mathbf{x}) = 1$ and $|f|, |\mathbf{x}| \leq p(\kappa)$, the following holds. There exist events A and B with $\Pr[A] = \Pr[B] = 1 - \epsilon_w(\kappa)$ such that*

$$\mathsf{Prove}_\kappa(f, \mathbf{x})|A \equiv \mathsf{Sim}(1^\kappa, f)|B.$$

Proof. Assume towards contradiction that the claim is not true, i.e., there exists a polynomial p such that for infinitely many κ's there exist a circuit f_κ with description-size at most $p(\kappa)$ and a satisfying assignment $\mathbf{x}_\kappa$, such that for every pair of events (A_κ, B_κ) with $\Pr[A] = \Pr[B] = 1 - \epsilon_w(\kappa)$ it holds that $\mathsf{Prove}_\kappa(f_\kappa, \mathbf{x}_\kappa)|A_\kappa \not\equiv \mathsf{Sim}(1^\kappa, f_\kappa)|B_\kappa$. Then, by Fact 8.6, for infinitely many κ it holds that $\mathsf{Prove}_\kappa(f_\kappa, \mathbf{x}_\kappa)$ is $\epsilon_w(\kappa)$-far from $\mathsf{Sim}(1^\kappa, f_\kappa)$, in contradiction to the $\epsilon_w(\kappa)$-statistical-WI of Π. This concludes the proof of the claim. □

Computational variant. We continue with the following computational variant of the indistinguishability hardcore lemma of [MT10], as appears in [BG24, Lemma 2.14]

Lemma 8.8 (Indistinguishability hardcore lemma)**.** *Let X and Y be two random variables, and let $\delta, \epsilon \in (0,1)$ and $s \in \mathbb{N}$ be given. Assume that for every s-sized distinguisher $\mathcal{D}$ it holds that*

$$\left| \Pr_{x \leftarrow X}[\mathcal{D}(x) = 1] - \Pr_{y \leftarrow Y}[\mathcal{D}(y) = 1] \right| \leq \delta.$$

Then there exists a pair of events A and B with probability $1-\delta$ each, so that for every s'-sized distinguisher $\mathcal{D}$ it holds that

$$\left| \Pr_{x \leftarrow X|A}[\mathcal{D}(x) = 1] - \Pr_{y \leftarrow Y|B}[\mathcal{D}(y) = 1] \right| \leq \epsilon,$$

where $s' = \frac{s \cdot \epsilon^2}{128(\log|\mathsf{Im}(X)| + \log|\mathsf{Im}(Y)| + 1)}$.

As an immediate corollary, we derive the following claim.

Claim 8.9. *Let $\Pi = (\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ be an NIP that satisfies $\epsilon_w(\kappa)$-WI and let Sim be the corresponding (inefficient) simulator. For every polynomials q, s and p there exists $\kappa_0 \in \mathbb{N}$ such that for every $\kappa \geq \kappa_0$, and every circuit f and assignment $\mathbf{x}$ that satisfy $f(\mathbf{x}) = 1$ and $|f|, |\mathbf{x}| \leq p(\kappa)$, the following holds. There exist events A and B with $\Pr[A] = \Pr[B] = 1 - \epsilon_w(\kappa)$ such that for every size-$s(\kappa)$ circuit $\mathcal{D}$ it holds that*

$$\left| \Pr_{\substack{r \leftarrow \mathcal{R}_\kappa^p(f,\mathbf{x})|A \\ (\mathsf{crs},\pi) = \mathsf{Prove}_\kappa(f,\mathbf{x};r)}}[\mathcal{D}(\mathsf{crs},\pi) = 1] - \Pr_{\substack{r \leftarrow \mathcal{R}_\kappa^s(f)|B \\ (\mathsf{crs},\pi) = \mathsf{Sim}(1^\kappa,f;r)}}[\mathcal{D}(\mathsf{crs},\pi) = 1] \right| \leq \frac{1}{q(\kappa)}.$$

Proof. Assume towards contradiction that there exist polynomials q, s and p for which the claim does not hold, i.e., for infinitely many κ's there exist a circuit f_κ with description-size at most $p(\kappa)$ and a satisfying assignment $\mathbf{x}_\kappa$, such that for every pair of events (A_κ, B_κ) with $\Pr[A_\kappa] = \Pr[B_\kappa] = 1 - \epsilon_w(\kappa)$ there exists a size-$s(\kappa)$ circuit $\mathcal{D}_\kappa$ so that

$$\left| \Pr_{\substack{r \leftarrow \mathcal{R}_\kappa^p(f_\kappa,\mathbf{x}_\kappa)|A_\kappa \\ (\mathsf{crs},\pi) = \mathsf{Prove}_\kappa(f_\kappa,\mathbf{x}_\kappa;r)}}[\mathcal{D}_\kappa(\mathsf{crs},\pi) = 1] - \Pr_{\substack{r \leftarrow \mathcal{R}_\kappa^s(f_\kappa)|B_\kappa \\ (\mathsf{crs},\pi) = \mathsf{Sim}(1^\kappa,f_\kappa;r)}}[\mathcal{D}_\kappa(\mathsf{crs},\pi) = 1] \right| > \frac{1}{q(\kappa)}.$$

Let us denote the set of those "bad" κ's by K. By the indistinguishability hardcore lemma (Lemma 8.8), for every such $\kappa \in K$ there exists a distinguisher $\mathcal{D}'_\kappa$ of size $s'(\kappa) = s(\kappa) \cdot 128(2\ell(\kappa) + 1) \cdot q(\kappa)^2 = \mathrm{poly}(\kappa)$ for which

$$\left| \Pr_{\substack{r \leftarrow \mathcal{R}_\kappa^p(f_\kappa,\mathbf{x}_\kappa) \\ (\mathsf{crs},\pi) = \mathsf{Prove}_\kappa(f_\kappa,\mathbf{x}_\kappa;r)}}[\mathcal{D}'_\kappa(1^\kappa,\mathsf{crs},\pi) = 1] - \Pr_{\substack{r \leftarrow \mathcal{R}_\kappa^s(f_\kappa) \\ (\mathsf{crs},\pi) = \mathsf{Sim}(1^\kappa,f_\kappa;r)}}[\mathcal{D}'_\kappa(1^\kappa,\mathsf{crs},\pi) = 1] \right| > \epsilon_w(\kappa),$$

where $\ell(\kappa)$ is a (polynomial) upper bound on the bit-length of (crs, π). Consider now the ensemble $\{f'_\kappa, \mathbf{x}'_\kappa\}_{\kappa \in \mathbb{N}}$ such that $(f'_\kappa, \mathbf{x}'_\kappa) = (f_\kappa, \mathbf{x}_\kappa)$ for all $\kappa \in K$, and otherwise $(f'_\kappa, \mathbf{x}'_\kappa)$ are set to be some arbitrary circuit and satisfying assignment of size at most $p(\kappa)$. Then Π does not satisfy ϵ_w-WI with respect to the ensemble $\{f'_\kappa, \mathbf{x}'_\kappa\}_{\kappa \in \mathbb{N}}$, in contradiction. This concludes the proof of the claim. □

Remark 8.10 (On measures, events, and efficient sampling)**.** *We mention that the original indistinguishability hardcore lemma of [MT10] uses the notion of measures. We consider here a simplified statement that uses events, and we observe that whenever we condition a variable X on an event A, we can think of X as a random variable over a refinement of the probability space in which the measure is translated naturally into an event. We also note that we can sample (possibly inefficiently) from X conditioned on A, by resampling from X until the event A occurs.*

8.1.4 Basic NIZK Transformations

In this section we present some basic NIZK transformations. The following lemma, due to [FLS90], shows that an NIP with $\epsilon_w(\kappa)$-WI can be strengthened to $(\epsilon_w(\kappa) + \mathsf{negl}(\kappa))$-zero knowledge. The lemma is proved in Section C.2.

Lemma 8.11. *Let $\tilde{\Pi} = (\tilde{\mathsf{Gen}}, \tilde{\mathcal{P}}, \tilde{\mathcal{V}})$ be an NIP that satisfies completeness, $\epsilon_s(\kappa)$-soundness and $\epsilon_w(\kappa)$-WI. Then, assuming the existence of a one-way function, there exists an NIP $\Pi = (\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ that satisfies completeness, $(\epsilon_s(\kappa) + \mathsf{negl}(\kappa))$-soundness and $(2\epsilon_w(\kappa) + \mathsf{negl}(\kappa))$-zero knowledge, where negl is some negligible function. In addition, the transformation preserves statistical soundness.*

The following lemma shows that we can transform statistical WI into statistical ZK, assuming that we have proof of knowledge. The lemma is proved in Section C.3.

Lemma 8.12. *Let $\tilde{\Pi} = (\tilde{\mathsf{Gen}}, \tilde{\mathcal{P}}, \tilde{\mathcal{V}})$ be an NIP that satisfies completeness, negl-proof of knowledge and $\epsilon_w(\kappa)$-statistical-WI. Then, assuming the existence of a statistically-hiding non-interactive commitment scheme, there exists an NIP $\Pi = (\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ that satisfies completeness, negl-proof of knowledge and $(2\epsilon_w(\kappa) + \mathsf{negl}(\kappa))$-statistical-zero-knowledge, where negl is some negligible function.*

The next lemma shows that an NIP with ϵ_s-soundness can be upgraded into an NIP with $(\epsilon_s + \mathsf{negl})$-proof of knowledge by using public-key encryption.The lemma is proved in Section C.4.

Lemma 8.13. *Let $\tilde{\Pi} = (\tilde{\mathsf{Gen}}, \tilde{\mathcal{P}}, \tilde{\mathcal{V}})$ be an NIP that satisfies completeness, $\epsilon_s(\kappa)$-soundness and $\epsilon_z(\kappa)$-zero knowledge. Then, assuming the existence of public-key encryption, there exists an NIP $\Pi = (\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ that satisfies completeness, $(\epsilon_s(\kappa) + \mathsf{negl}(\kappa))$-proof of knowledge and $(\epsilon_z(\kappa) + \mathsf{negl}(\kappa))$-zero knowledge, where negl is some negligible function.*

8.2 Main Amplifier: Negligible Soundness Error and Inverse-Polynomial WI Error

We continue by presenting our main amplifier.

Amplifier $\mathsf{Amplifier}_{\alpha,t,n,\Pi_{\mathsf{Commit}}}$

The amplifier is parameterized by functions $\alpha(\kappa)$, $t(\kappa)$ and $n(\kappa)$ that satisfy $2 \leq t(\kappa) \leq n(\kappa)$ for all $\kappa \in \mathbb{N}$. When κ is clear from the context we write $\alpha = \alpha(\kappa)$, $t = t(\kappa)$ and $n = n(\kappa)$. The amplifier is also parameterized by a commitment scheme $\Pi_{\mathsf{Commit}} = (\mathsf{Gen}_{\mathsf{Commit}}, \mathsf{Commit}, \mathsf{VOpen})$.

- **Building blocks:** The compiler C_{Th} promised in Corollary 7.5.
- **Inputs:** The amplifier takes as an input an NIP $\Pi_{wk} = (\mathsf{Gen}_{wk}, \mathcal{P}_{wk}, \mathcal{V}_{wk})$.
- **Output.** The amplifier outputs an NIP $\Pi = (\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ defined as follows.
  - The set-up algorithm $\mathsf{Gen}(1^\kappa)$ takes as an input a security parameter 1^κ. For every $i \in [n]$ it samples $\mathsf{crs}_i \leftarrow \mathsf{Gen}_{wk}(1^\kappa)$. In addition, it samples public parameters $\mathsf{pp} \leftarrow \mathsf{Gen}_{\mathsf{Commit}}(1^\kappa)$ for the commitment scheme, and outputs $\mathsf{crs} := (\mathsf{crs}_1, \ldots, \mathsf{crs}_n, \mathsf{pp})$.
  - The prover $\mathcal{P}(1^\kappa, \mathsf{crs}, f, \mathbf{x})$ is defined as follows.
    1. The prover takes as an input the security parameter 1^κ, a CRS $\mathsf{crs} = (\mathsf{crs}_1, \ldots, \mathsf{crs}_n, \mathsf{pp})$, a circuit-SAT instance f, and an assignment $\mathbf{x}$. Let $(R, W, \mathsf{Sim}_{\mathsf{LRNPSS}}, \mathsf{Dec}_{\mathsf{LRNPSS}}) = C_{\mathsf{Th}}(t, n)$.

2. The prover computes $(f_1, \ldots, f_n) = R(1^\alpha, f)$ and samples $(\mathbf{y}, \mathbf{y}_1, \ldots, \mathbf{y}_n) \leftarrow W(1^\alpha, f, \mathbf{x})$, where $\mathbf{y} \in \{0,1\}^\ell$.
3. For every $i \in [\ell]$ the prover commits to the i-th bit of $\mathbf{y}$, denoted $\mathbf{y}[i]$, i.e., $(C_i^y, o_i^y) \leftarrow \mathsf{Commit}_{\mathsf{pp}}(1^\kappa, \mathbf{y}[i])$. We denote $\mathbf{C}^y := (C_i^y)_{i\in[\ell]}$ and $\mathbf{O}^y := (o_i^y)_{i\in[\ell]}$.
4. For every $i \in [n]$, the prover computes the circuit $\hat{f}_i$ that (a) takes as an input a subset $I_i \subseteq [\ell]$, openings $(o_j)_{j\in I_i}$, and bits $(w_j)_{j\in I_i}$, (b) for every $j \in I_i$ verifies that $\mathsf{VOpen}_{\mathsf{pp}}(1^\kappa, C_j^y, o_j, w_j) = 1$, (c) defines the partial assignment $\mathbf{w}_i \in \{0,1,*\}^\ell$ by $\mathbf{w}_i[j] = w_j$ if $j \in I_i$ and $\mathbf{w}_i[j] = *$ otherwise, and (d) outputs 1 if the verification succeeded and the partial assignment $\mathbf{w}_i$ satisfies f_i, and outputs 0 otherwise. We observe that $\hat{f}_i$ is fully defined by f_i, pp and $\mathbf{C}^y$.
   For every $i \in [n]$, the prover also computes the assignment $\hat{\mathbf{y}}_i = (I_i, (o_j^y)_{j\in I_i}, (\mathbf{y}[j])_{j\in I_i})$, where $I_i := \{j \in [\ell] : \mathbf{y}_i[j] \neq *\}$.
5. For every $i \in [n]$ the prover samples $\pi_i \leftarrow \mathcal{P}_{wk}(1^\kappa, \mathsf{crs}_i, \hat{f}_i, \hat{\mathbf{y}}_i)$ and outputs $(\mathbf{C}^y, \pi_1, \ldots, \pi_n)$.

– The verifier $\mathcal{V}(1^\kappa, \mathsf{crs}, f, \pi)$ takes as an input a security parameter 1^κ, a CRS $\mathsf{crs} = (\mathsf{crs}_1, \ldots, \mathsf{crs}_n, \mathsf{pp})$, a circuit-SAT instance f and a proof $(\mathbf{C}^y, \pi_1, \ldots, \pi_n)$. Let $(R, W, \mathsf{Sim}_{\mathsf{LRNPSS}}, \mathsf{Dec}_{\mathsf{LRNPSS}}) = C_{\mathsf{Th}}(t, n)$.
  The verifier locally computes $\hat{f}_1, \ldots, \hat{f}_n$ by computing $(f_1, \ldots, f_n) = R(1^\alpha, f)$, and for every $i \in [n]$ the verifier computing $\hat{f}_i$ given f_i, pp and $\mathbf{C}^y$. For every $i \in [n]$, the verifier checks that $\mathcal{V}_{wk}(1^\kappa, \mathsf{crs}_i, \hat{f}_i, \pi_i) = 1$. If all the checks succeed the verifier outputs 1, and otherwise the verifier outputs 0.

Figure 18: Amplifier $\mathsf{Amplifier}_{\alpha,t,n,\Pi_{\mathsf{Commit}}}$

8.2.1 Amplification Lemmas

The following claims show that, with a proper choice of parameters and commitment schemes, our amplifier reduces the soundness error to negligible and the WI error to an arbitrarily small inverse polynomial.

Lemma 8.14 (Amplifying WI)**.** *Let Π_{wk} be an NIP that satisfies $\epsilon_w(\kappa)$-WI, where*

$$\epsilon_w(\kappa) \leq 1 - \frac{1}{\kappa^c} \tag{3}$$

for some constant c and all $\kappa \in \mathbb{N}$. Let $\delta_w(\kappa) = 1/\kappa^{c_w}$ where $c_w > 0$ is an arbitrary constant. Let $t \leq n$ and α be polynomially-bounded and efficiently-computable functions that satisfy

$$n(\kappa) \geq \left\lceil 50\kappa^{2c+1} \right\rceil, \quad t(\kappa) = \left\lceil (\epsilon_w(\kappa) + \frac{1}{10k^c})n \right\rceil + 1, \quad \text{and } \alpha(\kappa) \geq \max(\kappa^{c+3}, 100\kappa^{c_w}). \tag{4}$$

Then the following holds.

1. *Let Π_{Commit} be a statistically-hiding NICOM. Then the scheme $\Pi := \mathsf{Amplifier}_{\alpha,t,n,\Pi_{\mathsf{Commit}}}(\Pi_{wk})$ satisfies $\delta_w(\kappa)$-WI. In addition, if Π_{wk} satisfied $\epsilon_w(\kappa)$-Statistical-WI then Π satisfies $\delta_w(\kappa)$-Statistical-WI.*
2. *Let $\Pi_{\mathsf{TCom}} = (\mathsf{Gen}_{\mathsf{Commit}}, \mathsf{Gen}'_{\mathsf{Commit}}, \mathsf{Commit}, \mathsf{VOpen}, \mathsf{TCom})$ be a trapdoor commitment scheme, and let $\Pi_{\mathsf{Commit}} := (\mathsf{Gen}_{\mathsf{Commit}}, \mathsf{Commit}, \mathsf{VOpen})$ be the associated honest NICOM. Then the scheme $\Pi := \mathsf{Amplifier}_{\alpha,t,n,\Pi_{\mathsf{Commit}}}(\Pi_{wk})$ satisfies $\delta_w(\kappa)$-WI.*

We prove Lemma 8.14 in Section 8.2.2.

Lemma 8.15 (Amplifying proof of knowledge)**.** *Let Π_{wk} be an NIP that satisfies $\epsilon_p(\kappa)$ proof of knowledge, where*

$$\epsilon_p(\kappa) \leq 1 - \frac{1}{\kappa^c},$$

for some constant c and all $\kappa \in \mathbb{N}$. Let Π_{Commit} be a NICOM. Let $t \leq n$ and α be polynomially-bounded and efficiently-computable functions that satisfies $n(\kappa) \geq 70\kappa^{3c}(\kappa + \ln(125\kappa^{2c}))$ and $2 \leq t(\kappa) \leq (1 - \epsilon_p(\kappa) - \frac{4}{5\kappa^c})n$. Then the scheme $\Pi := \mathsf{Amplifier}_{\alpha,t,n,\Pi_{\mathsf{Commit}}}(\Pi_{wk})$ satisfies proof of knowledge (with negligible error probability).

We prove Lemma 8.15 in Section 8.2.3.

Lemma 8.16 (Amplifying statistical soundness)**.** *Let Π_{wk} be an NIP that satisfies $\epsilon_s(\kappa)$ statistical soundness, where*

$$\epsilon_s(\kappa) \leq 1 - \frac{1}{\kappa^c},$$

for some constant c and all $\kappa \in \mathbb{N}$. Let Π_{Commit} be a statistically-binding NICOM. Let $t \leq n$ and α be polynomially-bounded and efficiently-computable functions that satisfies $n(\kappa) \geq \kappa^{2c+1}$ and $2 \leq t(\kappa) \leq (1 - \epsilon_s(\kappa) - \frac{4}{5\kappa^c})n$. Then the scheme $\Pi := \mathsf{Amplifier}_{\alpha,t,n,\Pi_{\mathsf{Commit}}}(\Pi_{wk})$ satisfies statistical soundness (with negligible error probability).

We prove Lemma 8.16 in Section 8.2.4.

8.2.2 Proof of Lemma 8.14

First, let us assume that $\Pi_{\mathsf{Commit}} = (\mathsf{Gen}_{\mathsf{Commit}}, \mathsf{Commit}, \mathsf{VOpen})$ is statistically hiding and prove that computational-WI is amplified. We later show that with small modifications the same proof holds for statistical-WI, thus proving the first part of Lemma 8.14. Finally, we explain how to extend the proof to the case of trapdoor commitments, thus proving the second part of Lemma 8.14.

Notation. Let us recall the following notation. We denote the (possibly inefficient) simulator of Π_{wk} by Sim_{wk}. We denote by $\mathsf{Prove}_\kappa(f, \mathbf{x})$ the random variable (crs, π) where $\mathsf{crs} \leftarrow \mathsf{Gen}_{wk}(1^\kappa)$ and $\pi \leftarrow \mathcal{P}_{wk}(1^\kappa, \mathsf{crs}, f, \mathbf{x})$. We also denote the domain of randomness of $\mathsf{Prove}_\kappa(f, \mathbf{x})$ by $\mathcal{R}^p_\kappa(f, \mathbf{x})$ and the domain of randomness of $\mathsf{Sim}_{wk}(1^\kappa, f)$ by $\mathcal{R}^s_\kappa(f)$. We continue with the proof that proceeds in five steps.

1. **Step one: The new simulator.** We define the inefficient simulator Sim for Π in the following way. On input $(1^\kappa, f)$ the simulator first samples $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\kappa)$, finds in brute force the first assignment in lexicographic order $\mathbf{w}$ that satisfies f, samples $\pi \leftarrow \mathcal{P}(1^\kappa, \mathsf{crs}, f, \mathbf{w})$ and outputs (crs, π).

   Fix an ensemble $\{(f_\kappa, \mathbf{x}_\kappa)\}_{\kappa\in\mathbb{N}}$ and a polynomial p that satisfy $f_\kappa(\mathbf{x}_\kappa) = 1$ and $|f_\kappa|, |\mathbf{x}_\kappa| \leq p(\kappa)$ for every $\kappa \in \mathbb{N}$. For every κ let us denote by $\mathbf{w}_\kappa$ the first assignment in lexicographic order that satisfies f_κ, and assume towards contradiction that there exists a polynomial-sized non-uniform circuit family $\mathcal{D} = \{\mathcal{D}_\kappa\}_{\kappa\in\mathbb{N}}$, so that for infinitely many κ,

$$\left| \Pr_{\substack{\mathsf{crs}\leftarrow\mathsf{Gen}(1^\kappa) \\ \pi\leftarrow\mathcal{P}(1^\kappa,\mathsf{crs},f_\kappa,\mathbf{x}_\kappa)}} [\mathcal{D}_\kappa(1^\kappa, \mathsf{crs}, \pi) = 1] - \Pr_{\substack{\mathsf{crs}\leftarrow\mathsf{Gen}(1^\kappa) \\ \pi\leftarrow\mathcal{P}(1^\kappa,\mathsf{crs},f_\kappa,\mathbf{w}_\kappa)}} [\mathcal{D}_\kappa(1^\kappa, \mathsf{crs}, \pi) = 1] \right| > \delta_w(\kappa).$$

   Let us denote this set of "bad" κ's by K and the size of $\mathcal{D}$ by $s(\kappa)$.

2. **Step two: The experiments.** Let $\mathsf{genClaims}_\kappa(f, \mathbf{x})$ be the procedure that samples $\mathsf{pp} \leftarrow \mathsf{Gen}_{\mathsf{Commit}}(1^\kappa)$ and then follows Steps (2)–(4) of $\mathcal{P}$ in $\mathsf{Amplifier}_{\alpha,t,n,\Pi_{\mathsf{Commit}}}(\Pi_{wk})$ (see Figure 18) with security parameter 1^κ, the public parameters pp, the circuit f and the assignment $\mathbf{x}$ to obtain the values
$$((f_i, \mathbf{y}_i)_{i\in[n]}, \mathbf{y}, \mathbf{C}^y, \mathbf{O}^y, (\hat{f}_i, \hat{\mathbf{y}}_i)_{i\in[n]}).$$
We note that Steps (2)–(4) are independent of $\mathsf{crs}_1, \ldots, \mathsf{crs}_n$. The procedure outputs $(\mathsf{pp}, \mathbf{C}^y, (\hat{f}_i, \hat{\mathbf{y}}_i)_{i\in[n]})$.

We define the experiment $\mathsf{Exp}_\kappa(\mathbf{x})$ that outputs $(\mathsf{pp}, \mathsf{crs}_1, \ldots, \mathsf{crs}_n, \mathbf{C}^y, \pi_1, \ldots, \pi_n)$ where $(\mathsf{pp}, \mathbf{C}^y, (\hat{f}_i, \hat{\mathbf{y}}_i)_{i\in[n]})$ are sampled from $\mathsf{genClaims}_\kappa(f_\kappa, \mathbf{x})$, and (crs_i, π_i) is sampled by $(\mathsf{crs}_i, \pi_i) \leftarrow \mathsf{Prove}_\kappa(\hat{f}_i, \hat{\mathbf{y}}_i)$. We observe that $\mathsf{Exp}_\kappa(\mathbf{x}_\kappa)$ has the same distribution as the random variables $(\mathsf{pp}, \mathsf{crs}_1, \ldots, \mathsf{crs}_n, \mathbf{C}^y, \pi_1, \ldots, \pi_n)$ where $(\mathsf{pp}, \mathsf{crs}_1, \ldots, \mathsf{crs}_n) \leftarrow \mathsf{Gen}(1^\kappa)$, and $(\mathbf{C}^y, \pi_1, \ldots, \pi_n) \leftarrow \mathcal{P}(1^\kappa, \mathsf{crs}, f_\kappa, \mathbf{x}_\kappa)$. Similarly, $\mathsf{Exp}_\kappa(\mathbf{w}_\kappa)$ has the same distribution as $\mathsf{Sim}(1^\kappa, f_\kappa)$. Therefore, for every $\kappa \in K$, the adversary $\mathcal{D}$ distinguishes $\mathsf{Exp}_\kappa(\mathbf{x}_\kappa)$ from $\mathsf{Exp}_\kappa(\mathbf{w}_\kappa)$ with advantage greater than $\delta_w(\kappa)$.

3. **Step three: On good events.** Let p' be a polynomial bounding the description size of the circuits $\hat{f}_1, \ldots, \hat{f}_n$ and the assignments $\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_n$, and let $q(\kappa) := \delta_w(\kappa)/100n$. Claim 8.9, when applied on the polynomials (q, s, p') guarantees that there exists κ_0 such that for every $\kappa \geq \kappa_0$, every circuit F with description-size at most $p'(\kappa)$, and every satisfying assignment $\mathbf{m}$, there exist events $A_\kappa(F, \mathbf{m})$ and $B_\kappa(F, \mathbf{m})$ that satisfy $\Pr[A_\kappa(F, \mathbf{m})] = \Pr[B_\kappa(F, \mathbf{m})] = 1 - \epsilon_w(\kappa)$ such that for every circuit $\mathcal{D}'_\kappa$ of size $s(\kappa)$ it holds that
$$\left| \Pr_{\substack{r\leftarrow\mathcal{R}^p_\kappa(F,\mathbf{m})|A_\kappa(F,\mathbf{m}) \\ (\mathsf{crs},\pi)=\mathsf{Prove}_\kappa(F,\mathbf{m};r)}} [\mathcal{D}'_\kappa(1^\kappa, \mathsf{crs}, \pi) = 1] - \Pr_{\substack{r\leftarrow\mathcal{R}^s_\kappa(F)|B_\kappa(F,\mathbf{m}) \\ (\mathsf{crs},\pi)=\mathsf{Sim}_{wk}(1^\kappa,F;r)}} [\mathcal{D}'_\kappa(1^\kappa, \mathsf{crs}, \pi) = 1] \right| \leq \frac{1}{q(\kappa)}. \quad (5)$$

4. **Step four: Conditional experiments.** Let us denote the i-th instance of Prove_κ in Exp_κ by Prove^i_κ. Let A^i_κ be the event that $A_\kappa(\hat{f}_i, \hat{\mathbf{y}}_i)$ occurs in Prove^i_κ when Exp_κ picks the i-th circuit and assignment to be $(\hat{f}_i, \hat{\mathbf{y}}_i)$. We observe that $\Pr[A^i_\kappa] = 1 - \epsilon_w(\kappa)$ and that $A^1_\kappa, \ldots, A^n_\kappa$ are independent events. For a subset $I \subseteq [n]$ let A^I_κ be the event that A^i_κ occurs for every $i \in I$, and A^i_κ does not occur for $i \notin I$. We denote by $\mathsf{Exp}^I_\kappa(\mathbf{x})$ the experiment $\mathsf{Exp}_\kappa(\mathbf{x})$ conditioned on the event A^I_κ, i.e., in $\mathsf{Exp}^I_\kappa(\mathbf{x})$ we first sample $(\mathsf{pp}, \mathbf{C}^y, (\hat{f}_i, \hat{\mathbf{y}}_i)_{i\in[n]}) \leftarrow \mathsf{genClaims}_\kappa(f_\kappa, \mathbf{x})$ and then sample $(\mathsf{crs}_i, \pi_i) \leftarrow (\mathsf{Prove}^i_\kappa(\hat{f}_i, \hat{\mathbf{y}}_i) \mid A^i(\hat{f}_i, \hat{\mathbf{y}}_i))$ for all $i \in I$, and $(\mathsf{crs}_i, \pi_i) \leftarrow (\mathsf{Prove}^i_\kappa(\hat{f}_i, \hat{\mathbf{y}}_i) \mid \neg A^i(\hat{f}_i, \hat{\mathbf{y}}_i))$ for all $i \notin I$.

5. **Step five: Obtaining a contradiction.** Since $A^1_\kappa, \ldots, A^n_\kappa$ are independent and each event occurs with probability $1 - \epsilon_w(\kappa)$, and since $n \geq 50 \cdot \kappa^{2c+1}$, Chernoff's bound implies that the probability that A^I_κ occurs for I of size less than $n - t + 1$ is at most $e^{-\kappa}$. Therefore, for every $\kappa \in K$ there must exist some I of size at least $n - t + 1$ for which
$$\left|\Pr[\mathcal{D}_\kappa(\mathsf{Exp}^I_\kappa(\mathbf{x}_\kappa)) = 1] - \Pr[\mathcal{D}_\kappa(\mathsf{Exp}^I_\kappa(\mathbf{w}_\kappa)) = 1]\right| > \delta_w(\kappa) - e^{-\kappa},$$
where we used the fact that the probability that A^I_κ occurs in $\mathsf{Exp}_\kappa(\mathbf{x}_\kappa)$ is the same as the probability that A^I_κ occurs in $\mathsf{Exp}_\kappa(\mathbf{w}_\kappa)$.

The following key claim shows that $|\Pr[\mathcal{D}'_\kappa(\mathsf{Exp}^I_\kappa(\mathbf{x}_\kappa) = 1] - \Pr[\mathcal{D}'_\kappa(\mathsf{Exp}^I_\kappa(\mathbf{w}_\kappa) = 1]| < \delta_w(\kappa)/5$, which means that for infinitely many κ's it holds that $\delta_w(\kappa)/5 > \delta_w(\kappa) - e^{-\kappa}$, which is a contradiction since $\delta_w(\kappa)$ is inverse polynomial.

Claim 8.17. *For every $I \subseteq [n]$ of size at least $n - t + 1$, there exists an inefficient simulator Sim^I, so that for every $s(\kappa)$-sized non-uniform circuit family $\mathcal{D}' = \{\mathcal{D}'_\kappa\}_{\kappa\in\mathbb{N}}$, and all sufficiently large κ,*

$$|\Pr[\mathcal{D}'_\kappa(\mathsf{Exp}^I_\kappa(\mathbf{x}_\kappa) = 1] - \Pr[\mathcal{D}'_\kappa(\mathsf{Sim}^I(1^\kappa) = 1]| < \delta_w(\kappa)/10,$$

and

$$|\Pr[\mathcal{D}'_\kappa(\mathsf{Exp}^I_\kappa(\mathbf{w}_\kappa) = 1] - \Pr[\mathcal{D}'_\kappa(\mathsf{Sim}^I(1^\kappa) = 1]| < \delta_w(\kappa)/10.$$

Therefore,

$$|\Pr[\mathcal{D}'_\kappa(\mathsf{Exp}^I_\kappa(\mathbf{x}_\kappa) = 1] - \Pr[\mathcal{D}'_\kappa(\mathsf{Exp}^I_\kappa(\mathbf{w}_\kappa) = 1]| < \delta_w(\kappa)/5.$$

Proof. Let $Z := [n] \setminus I$ and observe that $|Z| \leq t - 1$. Let $\ell_r := \lceil \kappa/(1 - \epsilon_w(\kappa))\rceil$ and let F and $\mathbf{m}$ be a circuit and a satisfying assignment. For every $\mathbf{r} = (r_1, \ldots, r_{\ell_r})$ in the support of $(\mathcal{R}^s_\kappa(F))^{\ell_r}$, we define the leakage function $\eta^{\mathbf{r}}$ in the following way: The leakage function $\eta^{\mathbf{r}}$ takes as an input $(F, \mathbf{m})$ and outputs *the index* of the first string r_j such that the event $B_\kappa(F, \mathbf{m})$ occurs if the random string of $\mathsf{Sim}_{wk}(1^\kappa, F)$ is r_j. (If no such index exists then $\eta^{\mathbf{r}}$ outputs 0.) The output length of $\eta^{\mathbf{r}}$ is $\log(\ell_r) + 1 \leq \lfloor \log(\alpha(\kappa))\rfloor$ bits where the inequality follows from the definition of ℓ_r and Equations (3) and (4). In the following, we let $(R, W, \mathsf{Sim}_{\mathsf{LRNPSS}}, \mathsf{Dec}_{\mathsf{LRNPSS}}) = C_{\mathsf{Th}}(t, n)$. We continue with the definition of the simulator Sim^I.

Inputs: The simulator takes as an input a security parameter 1^κ.

Computation: The simulator does as follows.

(a) Samples $\mathsf{pp} \leftarrow \mathsf{Gen}_{\mathsf{Commit}}(1^\kappa)$ and computes $(f_1, \ldots, f_n) = R(1^{\alpha(\kappa)}, f_\kappa)$. Recall that the input-length of each f_i is denoted by ℓ.

(b) For every $i \in [\ell]$ the simulator commits to zero, i.e., $(C^y_i, o^0_i) \leftarrow \mathsf{Commit}_{\mathsf{pp}}(1^\kappa, 0)$. In addition, for every $i \in [\ell]$, the simulator samples (inefficiently) $(C, o) \leftarrow \mathsf{Commit}_{\mathsf{pp}}(1^\kappa, 1)$ conditioned on the event that $C = C^y_i$, and the simulator sets $o^1_i := o$. (Here we use the fact that the commitment scheme is statistically-hiding.) The simulator sets $\mathbf{C}^y := (C^y_i)_{i\in[\ell]}$, $\mathbf{O}^0 := (o^0_i)_{i\in[\ell]}$ and $\mathbf{O}^1 := (o^1_i)_{i\in[\ell]}$. For every $i \in [n]$ the simulator computes $\hat{f}_i$ given f_i, pp and $\mathbf{C}^y$.

(c) For every $i \in [n]$ and for every possible value of $\mathbf{r}$, define the leakage function $\tau^{\mathbf{r}}_i$ in the following way: (1) the leakage function takes as an input a partial assignment $\mathbf{w}_i \in \{0, 1, *\}^\ell$, and we denote the j-th bit of $\mathbf{w}_i$ by $\mathbf{w}_i[j]$, (2) the function computes $\hat{\mathbf{w}}_i := (I_i, (o^{\mathbf{w}_i[j]}_j)_{j\in I_i}, (\mathbf{w}_i[j])_{j\in I_i})$ where $I_i = \{j \in [\ell] : \mathbf{w}_i[j] \neq *\}$, (3) the function outputs $\eta^{\mathbf{r}}(\hat{f}_i, \hat{\mathbf{w}}_i)$.

(d) Recall that $\mathsf{Sim}_{\mathsf{LRNPSS}}$ is the simulator of the leakage-resilient NPSS scheme. The simulator executes $\mathsf{Sim}_{\mathsf{LRNPSS}}(1^\alpha, f, Z)$ to obtain $(\mathbf{w}_1, \ldots, \mathbf{w}_n)$.
For every $i \in Z$ the simulator computes $\hat{\mathbf{w}}_i := (I_i, (o^{\mathbf{w}_i[j]}_j)_{j\in I_i}, (\mathbf{w}_i[j])_{j\in I_i})$ where $I_i = \{j \in [\ell] : \mathbf{w}_i[j] \neq *\}$. The simulator samples $r \leftarrow \mathcal{R}^p_\kappa(\hat{f}_i, \hat{\mathbf{w}}_i) \mid (\neg A_\kappa(\hat{f}_i, \hat{\mathbf{w}}_i))$ and computes $(\mathsf{crs}_i, \pi_i) := \mathsf{Prove}_\kappa(\hat{f}_i, \hat{\mathbf{w}}_i; r)$.
For every $i \in I$ the simulator samples $\mathbf{r}^i \leftarrow (\mathcal{R}^s_\kappa(\hat{f}_i))^{\ell_r}$, computes the index $j_i := \tau^{\mathbf{r}^i}_i(\mathbf{w}_i)$, and computes $(\mathsf{crs}_i, \pi_i) := \mathsf{Sim}_{wk}(1^\kappa, \hat{f}_i; \mathbf{r}^i[j_i])$.

Outputs: The simulator outputs $\mathsf{pp}, \mathsf{crs}_1, \ldots, \mathsf{crs}_n, \mathbf{C}^y, \pi_1, \ldots, \pi_n$.

We continue by proving the first part of the claim. (The second part follows in a similar way.) To do so, we consider the following hybrids.

Hybrid 1. Hybrid 1 is simply $\mathsf{Exp}^I_\kappa(\mathbf{x}_\kappa)$.

Hybrid 2. In Hybrid 2 we sample $\mathsf{pp}, \mathbf{C}^y, (\hat{f}_i, \hat{\mathbf{y}}_i)_{i\in I}, (\mathsf{crs}_i, \pi_i)_{i\in Z}$ like in $\mathsf{Exp}^I_\kappa(\mathbf{x}_\kappa)$, but now for every $i \in I$ we sample (crs_i, π_i) in the following way: we sample $r \leftarrow \mathcal{R}^s_\kappa(\hat{f}_i) \mid B_\kappa(\hat{f}_i, \hat{\mathbf{y}}_i)$, and set $(\mathsf{crs}_i, \pi_i) = \mathsf{Sim}_{wk}(1^\kappa, \hat{f}_i; r)$. The output is $(\mathsf{pp}, \mathsf{crs}_1, \ldots, \mathsf{crs}_n, \mathbf{C}^y, \pi_1, \ldots, \pi_n)$.
For every security parameter $\kappa \geq \kappa_0$ and every size-$s(\kappa)$ circuit $\mathcal{D}'_\kappa$ it holds that $|\Pr[\mathcal{D}'_\kappa(\mathsf{hybrid}_1) = 1] - \Pr[\mathcal{D}'_\kappa(\mathsf{hybrid}_2) = 1]| \leq n/q(\kappa)$, since every adversary that violates the above can be transformed into an adversary that violates Equation 5.

Hybrid 3. In Hybrid 3 we sample $\mathsf{pp}, \mathbf{C}^y, (\hat{f}_i, \hat{\mathbf{y}}_i)_{i\in I}, (\mathsf{crs}_i, \pi_i)_{i\in Z}$ like in Hybrid 2, but now for every $i \in I$ we sample (crs_i, π_i) in the following way: we sample $\mathbf{r}^i = (r^i_1, \ldots, r^i_{\ell_r}) \leftarrow (\mathcal{R}^s_\kappa(\hat{f}_i))^{\ell_r}$, compute $j_i = \eta^{\mathbf{r}^i}(\hat{f}_i, \hat{\mathbf{y}}_i)$ and set $(\mathsf{crs}_i, \pi_i) = \mathsf{Sim}_{wk}(1^\kappa, \hat{f}_i; \mathbf{r}^i[j])$. The output is $(\mathsf{pp}, \mathsf{crs}_1, \ldots, \mathsf{crs}_n, \mathbf{C}^y, \pi_1, \ldots, \pi_n)$.
For every $i \in I$ we note that with probability at most at most $\epsilon_w(\kappa)^{\kappa/(1-\epsilon_w(\kappa))} = (1 - (1 - \epsilon_w(\kappa)))^{\kappa/(1-\epsilon_w(\kappa))} \leq e^{-\kappa}$ there is no j such that the event $B_\kappa(\hat{f}_i, \hat{\mathbf{y}}_i)$ occurs if the chosen random string of $\mathsf{Sim}_{wk}(1^\kappa, \hat{f})$ is $\mathbf{r}^i[j]$. In addition, conditioned on the event that such j exists, the string $\mathbf{r}^i[j]$ has the same distribution as $\mathcal{R}^s_\kappa(F_i) \mid B_\kappa(\hat{f}_i, \hat{\mathbf{y}}_i)$. We conclude that $\mathsf{hybrid}_2 \equiv_{\mathsf{negl}(\kappa)} \mathsf{hybrid}_3$.

Hybrid 4. In Hybrid 4 we sample $\mathsf{pp}, \mathbf{y}, (\mathbf{y}_i)_{i\in[n]}$ like in Hybrid 3. For every $i \in [\ell]$ we commit to zero $(C^y_j, o^0_j) \leftarrow \mathsf{Commit}_{\mathsf{pp}}(1^\kappa, 0)$. We set $\mathbf{C}^y := (C^y_i)_{i\in[\ell]}$. At this step we perform the following re-sampling of openings: For every $i \in [\ell]$ we sample again $(C, o) \leftarrow \mathsf{Commit}_{\mathsf{pp}}(1^\kappa, 1)$ conditioned on $C = C^y_i$, and we set $o^1_i := o$. For every $i \in [\ell]$ we define $o^y_i := o^{\mathbf{y}[i]}_i$. We set $\mathbf{O}^0 := (o^0_i)_{i\in[\ell]}$, $\mathbf{O}^1 := (o^1_i)_{i\in[\ell]}$ and $\mathbf{O}^y := (o^y_i)_{i\in[\ell]}$. For every $i \in [\ell]$ let $(\hat{f}_i, \hat{\mathbf{y}}_i)$ be the circuit and satisfying assignment defined by f_i, $\mathbf{y}_i$, pp, $\mathbf{C}^y$ and $\mathbf{O}^y$. We split into cases.

- For every $i \in Z$ we sample $r \leftarrow \mathcal{R}^p_\kappa(\hat{f}_i, \hat{\mathbf{y}}_i) \mid (\neg A_\kappa(\hat{f}_i, \hat{\mathbf{y}}_i))$ and compute $(\mathsf{crs}_i, \pi_i) := \mathsf{Prove}_\kappa(\hat{f}_i, \hat{\mathbf{y}}_i; r)$.
- For every $i \in I$ we (1) sample $\mathbf{r}^i = (r^i_1, \ldots, r^i_{\ell_r}) \leftarrow (\mathcal{R}^s_\kappa(\hat{f}_i))^{\ell_r}$, (2) compute $j = \tau^{\mathbf{r}^i}_i(\mathbf{y}_i)$, where τ is defined in Step (5c) of the simulator, and we note that whenever $\tau^{\mathbf{r}^i}_i$ requires the opening $o^{\mathbf{y}_i[j]}_j$ it uses the opening $o^y_j = o^{\mathbf{y}_i[j]}_j$, (3) finally, we then set $(\mathsf{crs}_i, \pi_i) = \mathsf{Sim}_{wk}(1^\kappa, \hat{f}_i; \mathbf{r}^i[j])$.

The output of the hybrid is $(\mathsf{pp}, \mathsf{crs}_1, \ldots, \mathsf{crs}_n, \mathbf{C}^y, \pi_1, \ldots, \pi_n)$.
We continue with an analysis. The random variables $\mathsf{pp}, \mathbf{y}, (\mathbf{y}_i)_{i\in[n]}$ have the same distribution as in Hybrid 3, and we fix them. Since the commitment is statistically-hiding, the random variables $\mathbf{C}^y$ in Hybrid 4 are $\mathsf{negl}(\kappa)$-close to the same random variables in Hybrid 3, and we fix them as well. Conditioned on those random variables, the openings $\mathbf{O}^y$ have the same distribution, and we fix them as well. This fully defines $\hat{f}_1, \ldots, \hat{f}_n$ and $\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_n$. for every $i \in Z$ we observe that (crs_i, π_i) is sampled from the same distribution as in Hybrid 3, and we fix those values. Finally, it is not hard to see that for every fixed $\mathbf{r}$ it holds that $\tau^{\mathbf{r}}_i(\mathbf{y}_i) = \eta^{\mathbf{r}}(\hat{f}_i, \hat{\mathbf{y}}_i)$. Therefore, the random variables $(\mathsf{crs}_i, \pi_i)_{i\in I}$ have the same distribution as in Hybrid 3. We conclude that $\mathsf{hybrid}_3 \equiv_{\mathsf{negl}(\kappa)} \mathsf{hybrid}_4$.

Hybrid 5. Hybrid 5 is the simulator $\mathsf{Sim}^I(1^\kappa)$. The random variables $\mathsf{pp}, \mathbf{C}^y, \mathbf{O}^0, \mathbf{O}^1$ have the same distribution as in Hybrid 4, and we fix them. Observe that $\hat{f}_1, \ldots, \hat{f}_n$ are fully defined given those values. We also observe that the random variables $(\mathbf{r}^i)_{i\in I}$ have the same distribution in both hybrids, and we fix them as well. This means that the leakage functions $(\tau^{\mathbf{r}^i}_i)_{i\in I}$ are the same in both hybrids.
By the privacy of the leakage-resilient NPSS it holds that

$$(\mathbf{y}_i)_{i\in Z}, (\tau^{\mathbf{r}^i}_i(\mathbf{y}_i))_{i\in I} \equiv_{1/\alpha} (\mathbf{w}_i)_{i\in Z}, (\tau^{\mathbf{r}^i}_i(\mathbf{w}_i))_{i\in I}$$

where $(\mathbf{y}_i)_{i\in[n]}$ are the random variables generated in hybrid_4 and $(\mathbf{w}_i)_{i\in[n]}$ are the random variables generated by $\mathsf{Sim}_{\mathsf{LRNPSS}}$ in Step (5d) of the simulator $\mathsf{Sim}^I(1^\kappa)$. Fix $(\mathbf{y}_i)_{i\in Z}, (\tau_i^{\mathbf{r}^i}(\mathbf{y}_i))_{i\in I}$ and $(\mathbf{w}_i)_{i\in Z}, (\tau_i^{\mathbf{r}^i}(\mathbf{w}_i))_{i\in I}$ as well, and observe that they fully define $(\hat{\mathbf{y}}_i)_{i\in Z}$. Finally, the output in both hybrids is obtained by applying the same deterministic procedure on the variables we've fixed so far. We conclude that Hybrid 4 is $(1/\alpha)$-close to Hybrid 5.

Since $\frac{n}{q(\kappa)} + \frac{1}{\alpha(\kappa)} + \mathsf{negl}(\kappa) < \delta_w(\kappa)/10$ for all sufficiently large κ, the claim follows. □

This concludes the proof of the lemma for the case of statistically-hiding commitments and computational-WI. We continue by explaining how to extend the proof to the case of statistical-WI.

On statistical-WI. Like before, we assume that Π_{Commit} is a statistically-hiding commitment scheme, but now we assume that Π_{wk} satisfies $\epsilon_w(\kappa)$-statistical-WI, and our goal is to prove that Π satisfies $\delta_w(\kappa)$-statistical-WI. We explain how to modify the original proof to the case of statistical-WI. In Step 1 of the proof ("The new simulator") the simulator is defined in the same way, and the only difference is that now the distinguisher $\mathcal{D}$ is of unbounded size. Step 2 of the proof ("The experiments") remains the same, and in Step 3 ("On good events"), instead of using Claim 8.9 we use the statistical version of the claim, i.e., Claim 8.7. In more details, Claim 8.7 when applied on the polynomial p' that bounds the description size of the circuits $\hat{f}_1, \ldots, \hat{f}_n$, guarantees that there exists κ_0 such that for all $\kappa \geq \kappa_0$, for every circuit F and satisfying assignment $\mathbf{m}$ with $|F|, |\mathbf{m}| \leq p'(\kappa)$ there are events $A_\kappa(F, \mathbf{m})$ and $B_\kappa(F, \mathbf{m})$ that occur with probability $1-\epsilon_w(\kappa)$ such that

$$\mathsf{Prove}_\kappa(F, \mathbf{m})|A_\kappa(F, \mathbf{m}) \equiv \mathsf{Sim}(1^\kappa, F)|B_\kappa(F, \mathbf{m}). \tag{6}$$

Step 4 ("Conditional experiments") remains the same, and in Step 5 ("Obtaining a contradiction") the only difference is in the proof of Claim 8.17 where hybrid_1 has the same distribution as hybrid_2 (or otherwise we obtain a contradiction to Equation 6). In particular, at the end of the proof of Claim 8.17 it suffices to require that $\frac{1}{\alpha(\kappa)} + \mathsf{negl}(\kappa) < \delta_w(\kappa)/10$. This concludes the case of statistical-WI.

On trapdoor commitments. We continue by explaining how to modify the proof for trapdoor commitments (and computational-WI). Let $\Pi_{\mathsf{TCom}} = (\mathsf{Gen}_{\mathsf{Commit}}, \mathsf{Gen}'_{\mathsf{Commit}}, \mathsf{Commit}, \mathsf{VOpen}, \mathsf{TCom})$ be a trapdoor commitment. Our goal is to prove that when the commitment scheme is $\Pi_{\mathsf{Commit}} = (\mathsf{Gen}_{\mathsf{Commit}}, \mathsf{Commit}, \mathsf{VOpen})$ then Π satisfies $\delta_w(\kappa)$-WI. The high level idea is to replace the honest set-up and commitment algorithms with the fake algorithms. By the properties of the trapdoor commitments, using the fake algorithms will be indistinguishable from using the honest algorithms. In addition, the fake algorithms provide perfect hiding, so we can follow the exact same proof as in the case of statistically-hiding commitments.

In more details Step 1 of the proof ("The new simulator") remains the same and defines the new simulator Sim, the ensemble $\{f_\kappa, \mathbf{x}_\kappa, \mathbf{w}_\kappa\}_{\kappa\in\mathbb{N}}$ and the polynomial-size distinguisher $\mathcal{D}$ that satisfies

$$\left| \Pr_{\substack{\mathsf{crs}\leftarrow\mathsf{Gen}(1^\kappa)\\ \pi\leftarrow\mathcal{P}(1^\kappa,\mathsf{crs},f_\kappa,\mathbf{x}_\kappa)}} [\mathcal{D}_\kappa(1^\kappa, \mathsf{crs}, \pi) = 1] - \Pr_{\substack{\mathsf{crs}\leftarrow\mathsf{Gen}(1^\kappa)\\ \pi\leftarrow\mathcal{P}(1^\kappa,\mathsf{crs},f_\kappa,\mathbf{w}_\kappa)}} [\mathcal{D}_\kappa(1^\kappa, \mathsf{crs}, \pi) = 1] \right| > \delta_w(\kappa)$$

for infinitely many κ's. For a circuit f and satisfying assignment $\mathbf{x}$, we formalize the use of the fake commitments by defining $\mathsf{hybrid}_\kappa(f,\mathbf{x})$ in the following way:

$\mathsf{hybrid}_\kappa(f,\mathbf{x})$

1. Sample $(\mathsf{pp},\tau)\leftarrow \mathsf{Gen}'_{\mathsf{Commit}}(1^\kappa)$, as well as $\mathsf{crs}_i\leftarrow \mathsf{Gen}_{wk}(1^\kappa)$ for all $i\in[n]$.
2. Let $(R,W,\mathsf{Sim}_{\mathsf{LRNPSS}},\mathsf{Dec}_{\mathsf{LRNPSS}}) = C_{\mathsf{Th}}(t,n)$, compute $(f_1,\ldots,f_n) = R(1^{\alpha(\kappa)},f)$ and sample $(\mathbf{y},\mathbf{y}_1,\ldots,\mathbf{y}_n)\leftarrow W(1^{\alpha(\kappa)},f,\mathbf{x})$, where $\mathbf{y}\in\{0,1\}^\ell$.
3. For every $i\in[\ell]$ sample $(C_i^y,o_{i,0},o_{i,1})\leftarrow \mathsf{TCom}_{\mathsf{pp}}(\tau)$ and set $o_i^y := o_{i,\mathbf{y}[i]}$. We denote $\mathbf{C}^y := (C_i^y)_{i\in[\ell]}$ and $\mathbf{O}^y := (o_i^y)_{i\in[\ell]}$.
4. For every $i\in[n]$, let $\hat{f}_i$ be the circuit that (1) takes as an input a subset $I_i\subseteq[\ell]$, openings $(o_j)_{j\in I_i}$, and bits $(w_j)_{j\in I_i}$, (2) for every $j\in I_i$ verifies that $\mathsf{VOpen}_{\mathsf{pp}}(1^\kappa,C_j^y,o_j,w_j)=1$, (3) defines the partial assignment $\mathbf{w}_i\in\{0,1,*\}^\ell$ by $\mathbf{w}_i[j]=w_j$ if $j\in I_i$ and $\mathbf{w}_i[j]=*$ otherwise, and (4) outputs 1 if the verification succeeded and the partial assignment $\mathbf{w}_i$ satisfies f_i, and outputs 0 otherwise. For every $i\in[n]$, define the assignment $\hat{\mathbf{y}}_i=(I_i,(o_j^y)_{j\in I_i},(\mathbf{y}[j])_{j\in I_i})$, where $I_i:=\{j\in[\ell]:\mathbf{y}_i[j]\neq *\}$.
5. For every $i\in[n]$ sample $\pi_i\leftarrow\mathcal{P}_{wk}(1^\kappa,\mathsf{crs}_i,\hat{f}_i,\hat{\mathbf{y}}_i)$.
6. Set $\mathsf{crs}=(\mathsf{crs}_1,\ldots,\mathsf{crs}_n,\mathsf{pp})$ and $\pi=(\mathbf{C}^y,\pi_1,\ldots,\pi_n)$ and output (crs,π).

Figure 19: $\mathsf{hybrid}_\kappa(f,\mathbf{x})$

By the indistinguishability of the real and fake mode, for all κ it holds that $\left|\Pr_{\mathsf{crs}\leftarrow\mathsf{Gen}(1^\kappa),\ \pi\leftarrow\mathcal{P}(1^\kappa,\mathsf{crs},f_\kappa,\mathbf{x}_\kappa)}[\mathcal{D}_\kappa(1^\kappa,\mathsf{crs},\pi)=1]-\Pr_{(\mathsf{crs},\pi)\leftarrow\mathsf{hybrid}_\kappa(f_\kappa,\mathbf{x}_\kappa)}[\mathcal{D}_\kappa(1^\kappa,\mathsf{crs},\pi)]\right| \le \mathsf{negl}(\kappa)$ and $\left|\Pr_{\mathsf{crs}\leftarrow\mathsf{Gen}(1^\kappa),\ \pi\leftarrow\mathcal{P}(1^\kappa,\mathsf{crs},f_\kappa,\mathbf{w}_\kappa)}[\mathcal{D}_\kappa(1^\kappa,\mathsf{crs},\pi)=1]-\Pr_{(\mathsf{crs},\pi)\leftarrow\mathsf{hybrid}_\kappa(f_\kappa,\mathbf{w}_\kappa)}[\mathcal{D}_\kappa(1^\kappa,\mathsf{crs},\pi)]\right| \le \mathsf{negl}(\kappa)$. Therefore,

$$\left|\Pr_{(\mathsf{crs},\pi)\leftarrow\mathsf{hybrid}_\kappa(f_\kappa,\mathbf{x}_\kappa)}[\mathcal{D}_\kappa(1^\kappa,\mathsf{crs},\pi)=1]-\Pr_{(\mathsf{crs},\pi)\leftarrow\mathsf{hybrid}(f_\kappa,\mathbf{w}_\kappa)}[\mathcal{D}_\kappa(1^\kappa,\mathsf{crs},\pi)=1]\right|>\delta_w(\kappa)-\mathsf{negl}(\kappa).$$

To get a contradiction, we follow the original proof except that we replace the honest commitment algorithms with the fake commitment algorithms. Formally, in Step 2 ("The experiments") we define $\mathsf{genClaims}_\kappa(f,\mathbf{x})$ to sample $(\mathsf{pp},\tau)\leftarrow\mathsf{Gen}'_{\mathsf{Commit}}(1^\kappa)$ follow Steps (2)–(4) of $\mathsf{hybrid}_\kappa(f,\mathbf{x})$ and output $(\mathsf{pp},\mathbf{C}^y,(\hat{f}_i,\hat{\mathbf{y}}_i)_{i\in[n]})$. Observe that now $\mathsf{Exp}_\kappa(\mathbf{x}_\kappa)$ has the same distribution as $\mathsf{hybrid}_\kappa(f_\kappa,\mathbf{x}_\kappa)$ and $\mathsf{Exp}_\kappa(\mathbf{w}_\kappa)$ has the same distribution as $\mathsf{hybrid}_\kappa(f_\kappa,\mathbf{w}_\kappa)$. Step 3 ("On good events") and Step 4 ("Conditional experiments") remain the same.

In Step 5 ("Obtaining a contradiction") the argument remains the same, except that throughout the proof, both in the definition of Sim^I and in the definitions of the hybrids, we use the fake commitment algorithms. In more details, we modify the simulator Sim^I in the following way: in Step (5a) we let the simulator sample $(\mathsf{pp},\tau)\leftarrow\mathsf{Gen}'_{\mathsf{Commit}}(1^\kappa)$ and in Step (5b) for every $i\in[\ell]$ we sample $(C_i^y,o_i^0,o_i^1)\leftarrow\mathsf{TCom}_{\mathsf{pp}}(\tau)$, and set $\mathbf{C}^y := (C_i^y)_{i\in[\ell]}$, $\mathbf{O}^0=(C_i^0)_{i\in[\ell]}$ and $\mathbf{O}^1=(C_i^1)_{i\in[\ell]}$. In addition, just like in Exp, in every hybrid we sample the trapdoor τ together with the public parameters, i.e., $(\mathsf{pp},\tau)\leftarrow\mathsf{Gen}'_{\mathsf{Commit}}(1^\kappa)$ (note that τ is never part of the hybrid's output, but only used to generate the fake commitments). Finally, in Hybrid 4 we sample we perform the re-sampling of the commitments in the same way we did in Sim^I, i.e., for every $i\in[\ell]$ we sample $(C_i^y,o_i^0,o_i^1)\leftarrow\mathsf{TCom}_{\mathsf{pp}}(\tau)$, and set $\mathbf{C}^y := (C_i^y)_{i\in[\ell]}$, $\mathbf{O}^0=(C_i^0)_{i\in[\ell]}$ and $\mathbf{O}^1=(C_i^1)_{i\in[\ell]}$. This concludes the analysis for trapdoor commitments and the proof of the lemma. □

8.2.3 Proof of Lemma 8.15

In this section we prove Lemma 8.15. Let us denote by Gen'_{wk} and Ext_{wk} the PPT algorithms promised by the proof of knowledge of Π_{wk}. In the following, we let $(R, W, \mathsf{Sim}_{\mathsf{LRNPSS}}, \mathsf{Dec}_{\mathsf{LRNPSS}}) = C_{\mathsf{Th}}(t, n)$. We define Gen' and Ext for the proof of knowledge of Π in the following way.

The key generator. $\mathsf{Gen}'(1^\kappa)$ takes as an input the security parameter 1^κ. For every $i \in [n]$ it samples $(\mathsf{crs}'_i, s'_i) \leftarrow \mathsf{Gen}'_{wk}(1^\kappa)$, and it also samples pp for the commitment scheme. It outputs $\mathsf{crs}' = (\mathsf{crs}'_1, \ldots, \mathsf{crs}'_n, \mathsf{pp})$ and $s' = (s'_1, \ldots, s'_n)$.

The extractor. The extractor $\mathsf{Ext}(1^\kappa, \mathsf{crs}', s', f, \pi)$ takes as an input the security parameter 1^κ, the CRS $\mathsf{crs}' = (\mathsf{crs}'_1, \ldots, \mathsf{crs}'_n, \mathsf{pp})$, the trapdoor $s' = (s'_1, \ldots, s'_n)$, a circuit f and a proof $\pi = (\mathbf{C}^y, \pi_1, \ldots, \pi_n)$. The extractor does as follows.

1. The extractor computes the functions $f_1, \ldots, f_n$, $\hat{f}_1, \ldots, \hat{f}_n$, given f, pp, and $\mathbf{C}^y$.
2. For every $i \in [n]$ the extractor executes $\mathsf{Ext}_{wk}(1^\kappa, \mathsf{crs}'_i, s'_i, \hat{f}_i, \pi_i)$ to obtain an assignment $\hat{\mathbf{y}}_i$. The extractor computes $I = \left\{ i \in [n] : \hat{f}_i(\hat{\mathbf{y}}_i) = 1 \right\}$, and for every $i \in I$ it parses $\hat{\mathbf{y}}_i = (I_i, (o_{i,j})_{j \in I_i}, (y_{i,j})_{j \in I_i})$. For every $i \in I$ the extractor defines the partial assignment $\mathbf{y}_i$ in the following way: for every $j \in I_i$ it sets $\mathbf{y}_i[j] := y_{i,j}$, and for every $j \notin I_i$ it sets $\mathbf{y}_i[j] := *$.
3. The extractor computes $\mathsf{Dec}_{\mathsf{LRNPSS}}(1^{\alpha(\kappa)}, f, I, (\mathbf{y}_i)_{i \in I})$ to obtain an assignment $\mathbf{x}$.
4. The extractor outputs $\mathbf{x}$.

Analysis. The CRS indistinguishability of $(\mathsf{Gen}'_{wk}, \mathsf{Ext}_{wk})$ implies that crs'_i has the same distribution as crs_i for every $i \in [n]$, and therefore crs' has the same distribution as crs. We continue with the extraction property.

Let $\mathcal{P}^* = \{\mathcal{P}^*_\kappa\}_{\kappa \in \mathbb{N}}$ be a polynomial-sized non-uniform circuit family of malicious provers, and let use denote the size of $\mathcal{P}^*$ by $s(\kappa)$. We assume without loss of generality that $\mathcal{P}^*$ always outputs a circuit f and a proof π such that $\mathcal{V}(1^\kappa, \mathsf{crs}', f, \pi) = 1$ (since if $\mathcal{V}$ outputs 0 then proof of knowledge is not violated). Consider the experiment Exp_κ in which we first sample $(\mathsf{crs}', s') \leftarrow \mathsf{Gen}'(1^\kappa)$, then compute $(f, \pi) \leftarrow \mathcal{P}^*_\kappa(\mathsf{crs}')$, and execute $\mathsf{Ext}(1^\kappa, \mathsf{crs}', s', f, \pi)$ to obtain $\mathbf{x}$. The experiment is successful if $f(\mathbf{x}) = 1$. Our goal is to show that the experiment succeeds with probability at least $1 - \mathsf{negl}(\kappa)$.

We say that proof of knowledge is violated in the i-th proof, if $\mathsf{Ext}_{wk}(1^\kappa, \mathsf{crs}'_i, s'_i, \hat{f}_i, \pi_i) = \hat{\mathbf{y}}_i$ and $\hat{f}_i(\hat{\mathbf{y}}_i) = 0$ and $\mathcal{V}_{wk}(1^\kappa, \mathsf{crs}'_i, \hat{f}_i, \pi_i) = 1$. Let E_κ be the event that for more than $(\epsilon_p(\kappa) + \frac{4}{5\kappa^c})$-fraction of the indices $i \in [n]$ it holds that proof of knowledge is violated in the i-th proof. The following claim bounds the probability of E_κ.

Claim 8.18. *It holds that* $\Pr[E_\kappa] \leq \mathsf{negl}(\kappa)$.

We postpone the proof of the claim to the end of the section. Let us condition on the event E_κ not occurring. Then, the set I computed by the extractor is of size at least $(1 - \epsilon_p(\kappa) - \frac{4}{5\kappa^c})n \geq t$. Let $\mathbf{O}$ be the set of all commitment-openings that Ext computes in Step (2). If there is a commitment $C \in \mathbf{C}^y$ and a pair of openings $o, o' \in \mathbf{O}$ such that $\mathsf{VOpen}_{\mathsf{pp}}(1^\kappa, C, o, 0) = \mathsf{VOpen}_{\mathsf{pp}}(1^\kappa, C, o', 1) = 1$ then the binding property is violated, and therefore this event occurs only with negligible probability, and we condition on this event not occurring. Then for every $i \in I$ it holds that (1) the

assignment $\hat{\mathbf{y}}_i$ satisfies $\hat{f}_i$, so $f_i(\mathbf{y}_i) = 1$ for all $i \in I$, and (2) the partial assignments $(\mathbf{y}_i)_{i\in I}$ are pairwise consistent on Boolean values. Therefore, by the perfect recovery of the leakage-resilient NPSS, the assignment $\mathbf{x}$ satisfies f, as required.

Finally, we continue with the proof of Claim 8.18.

Proof of Claim 8.18. The claim follows from the work of [IJK09] on direct product theorems. Formally, we consider *weakly verifiable puzzles* that consist of a pair of efficient algorithms $P = (G, V)$, where G is the puzzle-generator algorithm, that on security parameter 1^κ samples a puzzle x and some "check string" α, and V is the deterministic verifier that on security parameter 1^κ, puzzle and check string (x, α) and a solution y, outputs either 0 or 1. For every solver S, the success probability (with respect to κ) of S is the probability that $V(1^\kappa, x, \alpha, y) = 1$ where $(x, \alpha) \leftarrow G(1^\kappa)$ and $y \leftarrow S(x)$. We let the n-wise direct product, denoted P^n, be the puzzle that asks n independent challenges from G.

Following [IJK09], we say that a puzzle is $\delta(\kappa)$-hard for time $t(\kappa)$ if for every randomized algorithm running in time $t(\kappa)$, and for all sufficiently large κ, the success probability of the algorithm is at most $1-\delta(\kappa)$. We also say that the $n(\kappa)$-wise product is $\nu(\kappa)$-approximately $\delta(\kappa)$-hard for time $t(\kappa)$ if every randomized algorithm running in time $t(\kappa)$ has probability at least $\delta(\kappa)$ of answering incorrectly at least $\nu(\kappa)\cdot n(\kappa)$ of the $n(\kappa)$ challenges. We need the following version of Theorem 1.1 of [IJK09].[12]

Theorem 8.19 (Theorem 1.1 of [IJK09])**.** *Let κ be the security parameter, and let P be a weakly verifiable puzzle that is $\delta(\kappa)$-hard for time $t(\kappa)$, where $\delta(\kappa) = \Omega(1/\kappa^c)$ for some constant c. Let $n(\kappa)$ be some polynomial in κ, let $\gamma(\kappa) = \Omega(1/\kappa^c)$ for some constant c, and let*

$$\epsilon(\kappa) \geq (100/\gamma\delta)\cdot\exp(-\gamma^2\delta n/40). \tag{7}$$

Then the direct product puzzle P^n is $(1-\gamma)\delta$-approximately $(1-\epsilon)$-hard for time $t'(\kappa) = t(\kappa)\cdot \mathrm{poly}(\epsilon, 1/\kappa, n, \delta, \gamma) = t(\kappa)\cdot\mathrm{poly}(\epsilon, 1/\kappa)$.

In our context, the puzzle generator $G(1^\kappa)$ samples $(\mathsf{crs}', s') \leftarrow \mathsf{Gen}'_{wk}(1^\kappa)$ and outputs $x = \mathsf{crs}'$ and $\alpha = s'$. In addition, the verifier V takes 1^κ, $(x = \mathsf{crs}', \alpha = s')$ and $y = (f, \pi)$. The verifier V outputs 1 if proof of knowledge is violated, i.e., if $\mathsf{Ext}_{wk}(1^\kappa, \mathsf{crs}', s', f, \pi) = \mathbf{x}$ and $f(\mathbf{x}) = 0$ and $\mathcal{V}_{wk}(1^\kappa, \mathsf{crs}', f, \pi) = 1$. For every polynomial $p(\kappa)$, our puzzle (G, V) is $(1-\epsilon_p(\kappa))$-hard for time $p(\kappa)$ solvers, i.e., for every $p(\kappa)$-sized non-uniform family of circuits $\mathcal{A} = \{\mathcal{A}_\kappa\}_{\kappa\in\mathbb{N}}$, and for all sufficiently large κ, the success probability of $\mathcal{A}_\kappa$ is at most $\epsilon_p(\kappa)$.

Assume towards contradiction that the claim is not true, and for infinitely many κ's the event E_κ occurs with probability more than $1/q(\kappa)$ for some polynomial q. This implies that there exists an adversary $\mathcal{A} = \{\mathcal{A}_\kappa\}_{\kappa\in\mathbb{N}}$ of size $s'(\kappa) = s(\kappa) + \mathrm{poly}(\kappa)$ so that for infinitely many κ's, the probability that $\mathcal{A}_\kappa$ solves more than $(\epsilon_p(\kappa) + \frac{4}{5\kappa^c})$-fraction of the instances in the n-wise product of our puzzle is larger than $1/q(\kappa)$. Indeed, $\mathcal{A}_\kappa$ on $(\mathsf{crs}'_1, \ldots, \mathsf{crs}'_n)$ samples pp computes $(f, \pi) \leftarrow \mathcal{P}^*_\kappa(\mathsf{crs}'_1, \ldots, \mathsf{crs}'_n, \mathsf{pp})$, parses $\pi = (\mathbf{C}^y, \pi_1, \ldots, \pi_n)$, computes the functions $\hat{f}_1, \ldots, \hat{f}_n$ given pp, f, and $\mathbf{C}^y$, and outputs $(\hat{f}_i, \pi_i)_{i\in[n]}$. In the terminology of [IJK09], the n-wise product of the puzzle is not $(1-\epsilon_p(\kappa)-\frac{4}{5\kappa^c})$-approximately $(1-1/q(\kappa))$-hard for circuits of size $s'(\kappa)$. But then, Theorem 8.19 implies that for some polynomial $p(\kappa)$, our puzzle is not $(1-\epsilon_p(\kappa))$-hard for size-$p(\kappa)$ circuits, in contradiction.[13] This concludes the proof of the claim. □

[12] In the original version of the theorem, the parameters γ and δ were constants. In our version we allow δ and γ to be inverse polynomials in the security parameter κ, and we observe that the proof of [IJK09] captures this case as well.

[13] The parameters of Theorem 8.19 are assigned as follows: Their δ is our $1-\epsilon_p(\kappa)$, their n is our n, their γ is set to

8.2.4 Proof of Lemma 8.16

In this section we prove Lemma 8.16.

On good choice of public parameters. We first note that the statistical soundness of Π_{wk} implies that for every polynomial q and for all sufficiently large κ, with probability $1 - \epsilon_s(\kappa)$ over the choice of $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\kappa)$ there is no pair (f, π) of length $q(\kappa)$ such that f is not satisfiable and $\mathcal{V}(1^\kappa, \mathsf{crs}, f, \pi) = 1$, or otherwise an unbounded adversary with output-length $q(\kappa)$ could violate the statistical soundness with non-negligible probability by iterating over all pairs (f, π) of length $q(\kappa)$. We call such a CRS *good* with respect to q.

Analyzing statistical soundness. Let $\mathcal{P}^* = \{\mathcal{P}^*_\kappa\}_{\kappa\in\mathbb{N}}$ be a family of (unbounded-size) non-uniform family of provers with polynomial output length q. Consider the experiment where we sample $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\kappa)$ and we let $(f, \pi) \leftarrow \mathcal{P}^*_\kappa(\mathsf{crs})$. Let us assume without loss of generality that $\mathcal{P}^*$ always outputs a circuit f that is not satisfiable. We say that $\mathcal{P}^*$ wins if $\mathcal{V}(1^\kappa, \mathsf{crs}, f, \pi) = 1$. That is, if we denote $\pi = (\mathbf{C}^y, \pi_1, \ldots, \pi_n)$ and set $\hat{f}_1, \ldots, \hat{f}_n$ to be the circuits defined by f, pp and $\mathbf{C}^y$, then $\mathcal{P}^*$ wins if $\mathcal{V}_{wk}(1^\kappa, \mathsf{crs}_i, \hat{f}_i, \pi_i) = 1$ for all $i \in [n]$. Our goal is to prove that $\mathcal{P}^*$ wins with negligible probability.

First, we observe that since f has description of length at most $q(\kappa)$, then there is some polynomial $q'(\kappa)$ such that $\hat{f}_1, \ldots, \hat{f}_n$ have description of length at most $q'(\kappa)$. For a sufficiently large κ,, Chernoff bound implies that with probability at least $1 - \exp(-\kappa)$, at most $(\epsilon_s(\kappa) + \frac{4}{5\kappa^c})n$ of the strings $\mathsf{crs}_1, \ldots, \mathsf{crs}_n$ are not good with respect to $q'(\kappa)$. Let us condition on this event, and denote by $I \subseteq [n]$ the set of all indices $i \in [n]$ such that crs_i is good with respect to $q'(\kappa)$, and observe that $|I| \geq (1 - \epsilon_s(\kappa) - \frac{4}{5\kappa^c})n \geq t$.

Consider any tuple $(f, \pi = (\mathbf{C}^y, \pi_1, \ldots, \pi_n))$ such that $\mathcal{V}_{wk}(1^\kappa, \mathsf{crs}_i, \hat{f}_i, \pi_i) = 1$ for all $i \in [n]$. First, since for every $i \in I$ it holds that crs_i is good with respect to q', then $\hat{f}_i$ is satisfiable. We let $\hat{\mathbf{y}}_i := (I_i, (o^i_j)_{j\in I_i}, (y^i_j)_{j\in I_i})$ be some satisfying assignment, we observe that $y^i_j \in \{0, 1\}$ for all $j \in I_i$, and we denote by $\mathbf{y}_i$ be the corresponding partial assignment, i.e., $\mathbf{y}_i[j] = y^i_j$ if $j \in I_i$ and $\mathbf{y}_i[j] = *$ otherwise. We also denote by $\mathbf{O}$ all the commitments that appear in those assignments. If there is a commitment $C \in \mathbf{C}^y$ and two openings $o, o' \in \mathbf{O}$ such that $\mathsf{VOpen}(C, o, 0) = \mathsf{VOpen}(C, o, 1) = 1$ then the statistical binding property is violated, and therefore this event occurs only with negligible probability. Let us condition on this event not occurring. Then it must hold that that partial assignments $(\mathbf{y}_i)_{i\in I}$ agree on Boolean values. In addition, $\mathbf{y}_i$ satisfies f_i for all $i \in I$ since $\hat{\mathbf{y}}_i$ satisfies $\hat{f}_i$. Now the perfect recovery of the leakage resilient NPSS implies that f is satisfiable. This concludes the proof of the lemma. □

8.3 From Inverse-Polynomial to Negligible WI Error

In this section we explain how to amplify a NIP with a sufficiently-small inverse-polynomial WI error ϵ_w and negligible soundness error ϵ_s, into a NIP with negligible WI error. Our amplifier relies on the following construction of (non-leakage-resilient) NPSS, taken from [AK25].

$\frac{4}{5\kappa^c}$, their ϵ is $1/q(\kappa)$ and their t' is $s'(\kappa)$. One can verify that by our choice of n, the condition in the theorem is satisfied. Indeed, the LHS of Equation 7 is inverse polynomial $\epsilon = 1/q(\kappa)$, while the RHS of Equation 7 is negligible and bounded by $\exp(-\kappa)$, and therefore Equation 7 is satisfied for all sufficiently large κ.

Theorem 8.20 ([AK25]). *There is a deterministic compiler C_{Th} that takes as an input two integers k and n, runs in time $\mathrm{poly}(n)$ and outputs the description of a k-out-of-n NPSS $(R, W, \mathsf{Sim}, \mathsf{Dec})$. The running time of all algorithms in $(R, W, \mathsf{Sim}, \mathsf{Dec})$ is $\mathrm{poly}(n \cdot |f|)$ where f is the circuit-SAT instance that the algorithms take as an input, and $|f|$ is the circuit-size of f measured by the number of wires.*

We continue by presenting the amplifier.

Amplifier $\mathsf{Amplifier}^{WI}_{n,\Pi_{\mathsf{Commit}}}$

The amplifier is parameterized by a function $n(\kappa)$ and a non-interactive commitment scheme $\Pi_{\mathsf{Commit}} = (\mathsf{Gen}_{\mathsf{Commit}}, \mathsf{Commit}, \mathsf{VOpen})$.

- **Building blocks:** An n-out-of-n NPSS $(R, W, \mathsf{Sim}_{\mathsf{NPSS}}, \mathsf{Dec}_{\mathsf{NPSS}})$, promised by Theorem 8.20.
- **Inputs:** The amplifier takes as an input an NIP $\Pi_{wk} = (\mathsf{Gen}_{wk}, \mathcal{P}_{wk}, \mathcal{V}_{wk})$.
- **Output.** The amplifier outputs an NIP $\Pi = (\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ defined as follows.
  - The set-up algorithm $\mathsf{Gen}(1^\kappa)$ takes as an input a security parameter 1^κ. For every $i \in [n]$ it samples $\mathsf{crs}_i \leftarrow \mathsf{Gen}_{wk}(1^\kappa)$. In addition, it samples public parameters $\mathsf{pp} \leftarrow \mathsf{Gen}_{\mathsf{Commit}}(1^\kappa)$ for the commitment scheme, and outputs $\mathsf{crs} := (\mathsf{crs}_1, \ldots, \mathsf{crs}_n, \mathsf{pp})$.
  - The prover $\mathcal{P}(1^\kappa, \mathsf{crs}, f, \mathbf{x})$ is defined as follows.
    1. The prover takes as an input the security parameter 1^κ, a CRS $\mathsf{crs} = (\mathsf{crs}_1, \ldots, \mathsf{crs}_n, \mathsf{pp})$, a circuit-SAT instance f, and an assignment $\mathbf{x}$. Let $(R, W, \mathsf{Sim}_{\mathsf{NPSS}}, \mathsf{Dec}_{\mathsf{NPSS}}) = C_{\mathsf{Th}}(n, n)$.
    2. The prover computes $(f_1, \ldots, f_n) = R(f)$ and samples $(\mathbf{y}, \mathbf{y}_1, \ldots, \mathbf{y}_n) \leftarrow W(f, \mathbf{x})$, where $\mathbf{y} \in \{0,1\}^\ell$.
    3. For every $i \in [\ell]$ the prover commits to the i-th bit of $\mathbf{y}$, denoted $\mathbf{y}[i]$, i.e., $(C^y_i, o^y_i) \leftarrow \mathsf{Commit}_{\mathsf{pp}}(1^\kappa, \mathbf{y}[i])$. We denote $\mathbf{C}^y := (C^y_i)_{i\in[\ell]}$ and $\mathbf{O}^y := (o^y_i)_{i\in[\ell]}$.
    4. For every $i \in [n]$, the prover computes the circuit $\hat{f}_i$ that (1) takes as an input a subset $I_i \subseteq [\ell]$, openings $(o_j)_{j\in I_i}$, and bits $(w_j)_{j\in I_i}$, (2) for every $j \in I_i$ verifies that $\mathsf{VOpen}_{\mathsf{pp}}(1^\kappa, C^y_j, o_j, w_j) = 1$, (3) defines the partial assignment $\mathbf{w}_i \in \{0,1,*\}^\ell$ by $\mathbf{w}_i[j] = w_j$ if $j \in I_i$ and $\mathbf{w}_i[j] = *$ otherwise, and (4) outputs 1 if the verification succeeded and the partial assignment $\mathbf{w}_i$ satisfies f_i, and outputs 0 otherwise. We observe that $\hat{f}_i$ is fully defined by f_i, pp and $\mathbf{C}^y$.
       For every $i \in [n]$, the prover also computes the assignment $\hat{\mathbf{y}}_i = (I_i, (o^y_j)_{j\in I_i}, (\mathbf{y}[j])_{j\in I_i})$, where $I_i := \{j \in [\ell] : \mathbf{y}_i[j] \neq *\}$.
    5. For every $i \in [n]$ the prover samples $\pi_i \leftarrow \mathcal{P}_{wk}(1^\kappa, \mathsf{crs}_i, \hat{f}_i, \hat{\mathbf{y}}_i)$ and outputs $(\mathbf{C}^y, \pi_1, \ldots, \pi_n)$.
  - The verifier $\mathcal{V}(1^\kappa, \mathsf{crs}, f, \pi)$ takes as an input a security parameter 1^κ, a CRS $\mathsf{crs} = (\mathsf{crs}_1, \ldots, \mathsf{crs}_n, \mathsf{pp})$, a circuit-SAT instance f and a proof $(\mathbf{C}^y, \pi_1, \ldots, \pi_n)$.
    The verifier locally computes $\hat{f}_1, \ldots, \hat{f}_n$ by computing $(f_1, \ldots, f_n) = R(f)$, and for every $i \in [n]$ the verifier computing $\hat{f}_i$ given f_i, pp and $\mathbf{C}^y$. For every $i \in [n]$, the verifier checks that $\mathcal{V}_{wk}(1^\kappa, \mathsf{crs}_i, \hat{f}_i, \pi_i) = 1$. If all the checks succeed the verifier outputs 1, and otherwise the verifier outputs 0.

Figure 20: Amplifier $\mathsf{Amplifier}^{WI}_{n,\Pi_{\mathsf{Commit}}}$

The following lemma shows that WI is indeed amplified.

Lemma 8.21 (WI amplification). *Let Π_{wk} be an NIP that satisfies ϵ_w-WI where $\epsilon_w(\kappa) = 1/\kappa$, and let $n(\kappa) = \lfloor \log(\kappa) \rfloor$. Then the following holds.*

- *If* Π_{Commit} *is a statistically-hiding NICOM, then the scheme* $\Pi := \mathsf{Amplifier}^{WI}_{n,\Pi_{\mathsf{Commit}}}(\Pi_{wk})$ *satisfies WI (with negligible error probability). In addition, if* Π_{wk} *satisfied* $\epsilon_w(\kappa)$-statistical-*WI, then* Π *satisfies statistical-WI.*

- *If* $\Pi_{\mathsf{TCom}} = (\mathsf{Gen}_{\mathsf{Commit}}, \mathsf{Gen}'_{\mathsf{Commit}}, \mathsf{Commit}, \mathsf{VOpen}, \mathsf{TCom})$ *is a trapdoor NICOM, and* $\Pi_{\mathsf{Commit}} := (\mathsf{Gen}_{\mathsf{Commit}}, \mathsf{Commit}, \mathsf{VOpen})$ *is the associated honest NICOM, then the scheme* $\Pi := \mathsf{Amplifier}^{WI}_{n,\Pi_{\mathsf{Commit}}}(\Pi_{wk})$ *satisfies WI (with negligible error probability).*

We prove Lemma 8.21 in Section 8.3.1. We also state the following lemma that shows that the transformation preserves statistical soundness or proof-of-knowledge. The proof of the lemma is similar to that of Lemma 8.15 or Lemma 8.16, and is therefore omitted.

Lemma 8.22 (Preserving statistical-soundness/proof-of-knowledge.)**.** *Let* Π_{wk} *be an NIP. Let* $n(\kappa)$ *be polynomially-bounded and efficiently computable. Then the following holds.*

- *If* Π_{wk} *satisfies proof of knowledge and* Π_{Commit} *is a NICOM, then the scheme* $\Pi := \mathsf{Amplifier}^{WI}_{n,\Pi_{\mathsf{Commit}}}(\Pi_{wk})$ *satisfies proof-of-knowledge.*

- *If* Π_{wk} *satisfies statistical soundness, and* $\Pi_{\mathsf{Commit}} := (\mathsf{Gen}_{\mathsf{Commit}}, \mathsf{Commit}, \mathsf{VOpen})$ *is a statistically-binding NICOM, then the scheme* $\Pi := \mathsf{Amplifier}^{WI}_{n,\Pi_{\mathsf{Commit}}}(\Pi_{wk})$ *satisfies statistical soundness.*

8.3.1 Proof of Lemma 8.21

Our analysis is closely related to the amplification result of [BKP+24, Claim 4.18], which, in turn, is a variant of the results of [LM20]. While the original works are limited to the statistical setting, we show that the argument can be simplified and extended to the computational setting via the use of the XOR lemma.

Recall that the lemma distinguishes between two cases: (1) amplification of *statistical*-WI via the use of statistically-hiding commitments; and (2) amplification of *computational*-WI with either statistically-hiding commitments or trapdoor commitments. We will try to handle these case in a unified way, and split the argument into cases only when this is necessary. We begin with the following notation.

Notation. Let Sim_{wk} be the (inefficient) simulator of Π_{wk}, and let $\mathsf{Prove}_\kappa(f, \mathbf{x})$ be the efficient procedure that given $(f, \mathbf{x})$ samples $\mathsf{crs} \leftarrow \mathsf{Gen}_{wk}(1^\kappa)$ and $\pi \leftarrow \mathcal{P}_{wk}(1^\kappa, \mathsf{crs}, f, \mathbf{x})$ and outputs (crs, π). For two algorithms A and B, and a bit $b \in \{0,1\}$, we denote by $F := \langle A, B\rangle_b$ the algorithm that is equal to A if $b = 0$, and is equal to B if $b = 1$.

The new simulator. We define the inefficient simulator Sim for Π in the following way. On input $(1^\kappa, f)$ the simulator first samples $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\kappa)$, finds in brute force the first assignment in lexicographic order $\mathbf{w}$ that satisfies f, samples $\pi \leftarrow \mathcal{P}(1^\kappa, \mathsf{crs}, f, \mathbf{w})$ and outputs (crs, π). To prove that WI holds, we reduce WI to the XOR lemma (see Section 8.1.2).

Hybrid combiner $\mathcal{C}'$. For the analysis, it will be useful to consider the following "hybrid-combiner" $\mathcal{C}'_{\mathbf{s}}(1^\kappa, f, \mathbf{x})$ that is parameterized by a string $\mathbf{s} \in \{0,1\}^n$ and takes as an input a security parameter 1^κ, a circuit-SAT instance f and a satisfying assignment $\mathbf{x}$. On input $(1^\kappa, f, \mathbf{x})$ we let $\mathcal{C}'_{\mathbf{s}}$ sample $\mathsf{pp} \leftarrow \mathsf{Gen}_{\mathsf{Commit}}(1^\kappa)$, compute $(f_1, \ldots, f_n) = R(f)$ and sample $(\mathbf{y}, \mathbf{y}_1, \ldots, \mathbf{y}_n) \leftarrow$

$W(f, \mathbf{x})$, where $\mathbf{y} \in \{0,1\}^\ell$. Then, $\mathcal{C}_\mathbf{s}$ commits to the i-th bit of $\mathbf{y}$, denoted $\mathbf{y}[i]$, i.e., $(C_i^y, o_i^y) \leftarrow \mathsf{Commit}_{\mathsf{pp}}(1^\kappa, \mathbf{y}[i])$. We denote $\mathbf{C}^y := (C_i^y)_{i\in[\ell]}$ and $\mathbf{O}^y := (o_i^y)_{i\in[\ell]}$, and let $\hat{f}_i, \hat{\mathbf{y}}_i$ be the circuit and satisfying assignment defined by $\mathsf{pp}, f_i, \mathbf{y}_i, \mathbf{C}^y$ and $\mathbf{O}^y$. Then for every $i \in [n]$, if $s_i = 0$ then $\mathcal{C}'_\mathbf{s}$ samples $(\mathsf{crs}_i, \pi_i) \leftarrow \mathsf{Prove}_\kappa(\hat{f}_i, \hat{\mathbf{y}}_i)$, and if $s_i = 1$ then $\mathcal{C}'_\mathbf{s}$ samples $(\mathsf{crs}_i, \pi_i) \leftarrow \mathsf{Sim}_{wk}(1^\kappa, \hat{f}_i)$. Finally $\mathcal{C}'_\mathbf{s}$ outputs $(\mathsf{pp}, \mathsf{crs}_1, \ldots, \mathsf{crs}_n, \mathbf{C}^y, \pi_1, \ldots, \pi_n)$.

Hybrid combiner $\mathcal{C}$. We continue by defining the hybrid combiner $\mathcal{C}$ that is only required for the case of trapdoor commitments. However, in order to unify notation, if the commitment scheme is statistically hiding, then we take $\mathcal{C}_\mathbf{s}$ to be the same as $\mathcal{C}'_\mathbf{s}$. In the case of a trapdoor commitment scheme, $\mathcal{C}_\mathbf{s}$ is defined like $\mathcal{C}'_\mathbf{s}$, but with respect to the "fake" mode of the trapdoor commitment scheme. That is, $\mathcal{C}_\mathbf{s}(1^\kappa, f, \mathbf{x})$ follows the same steps of $\mathcal{C}'_\mathbf{s}(1^\kappa, f, \mathbf{x})$ except that (1) the public parameters of the commitments scheme are sampled from the fake set-up algorithm together with the trapdoor, $(\mathsf{pp}, \tau) \leftarrow \mathsf{Gen}'_{\mathsf{Commit}}(1^\kappa)$, and (2) the commitment of the i-th bit of $\mathbf{y}$ is computed by first sampling $(C_i^y, o_i^0, o_i^1) \leftarrow \mathsf{TCom}(1^\kappa, \tau)$, setting $o_i^y := o_i^{\mathbf{y}[i]}$, and then setting the i-th commitment and opening to be (C_i^y, o_i^y). Observe that in the case of trapdoor commitment, for every polynomially-bounded ensemble $\{f'_\kappa, \mathbf{x}'_\kappa\}_{\kappa\in\mathbb{N}}$ it holds that $\mathcal{C}_\mathbf{s}(1^\kappa, f'_\kappa, \mathbf{x}'_\kappa)$ is computationally indistinguishable from $\mathcal{C}'_\mathbf{s}(1^\kappa, f'_\kappa, \mathbf{x}'_\kappa)$.

Analysis. Consider any ensemble $\{f_\kappa, \mathbf{x}_\kappa^0\}_{\kappa\in\mathbb{N}}$ of polynomially-bounded circuit f_κ and a satisfying assignment $\mathbf{x}_\kappa$. For every κ, let us denote by $\mathbf{x}_\kappa^1$ the first assignment in lexicographic order that satisfies f_κ, and observe that this is the assignment that Sim finds. In the case of statistical-WI with statistically-hiding commitments, it suffices to prove that there is a negligible function ν such that for every κ it holds that

$$\mathcal{C}_{0^n}(1^\kappa, f_\kappa, \mathbf{x}_\kappa^0) \equiv_{\nu(\kappa)} \mathcal{C}_{0^n}(1^\kappa, f_\kappa, \mathbf{x}_\kappa^1),$$

and in the case of computational-WI, it suffices to prove that

$$\mathcal{C}_{0^n}(1^\kappa, f_\kappa, \mathbf{x}_\kappa^0) \approx \mathcal{C}_{0^n}(1^\kappa, f_\kappa, \mathbf{x}_\kappa^1).$$

For $\mathbf{s} \in \{0,1\}^n$ let us denote $H_{\kappa,\mathbf{s}} := \mathcal{C}_\mathbf{s}(1^\kappa, f_\kappa, \mathbf{x}_\kappa^0)$ and $H'_{\kappa,\mathbf{s}} := \mathcal{C}_\mathbf{s}(1^\kappa, f_\kappa, \mathbf{x}_\kappa^1)$. Let ζ_n (resp., Σ_n) be the uniform distribution over all strings in $\{0,1\}^n$ with even (resp., odd) Hamming weight. For an adversary $\mathcal{A}$ and two random variables X and Y, let us denote $\Delta_\mathcal{A}(X, Y) := |\Pr[\mathcal{A}(X) = 1] - \Pr[\mathcal{A}(Y) = 1]|$. We continue with the following claim.

Claim 8.23. *For every distinguisher $\mathcal{A} = \{\mathcal{A}_\kappa\}_{\kappa\in\mathbb{N}}$ and every $\kappa \in \mathbb{N}$ it holds that*

$$\Delta_{\mathcal{A}_\kappa}(H_{\kappa,0^n}, H'_{\kappa,0^n}) \leq 2^{n-1} \cdot \left[\Delta_{\mathcal{A}_\kappa}(H_{\kappa,\zeta_n}, H_{\kappa,\Sigma_n}) + \Delta_{\mathcal{A}_\kappa}(H'_{\kappa,\zeta_n}, H'_{\kappa,\Sigma_n}) + 2 \cdot \max_{\mathbf{s}\in\{0,1\}^n\setminus\{0^n\}} \Delta_{\mathcal{A}_\kappa}(H_{\kappa,\mathbf{s}}, H'_{\kappa,\mathbf{s}})\right].$$

Proof. It holds that

$$\begin{aligned}
\Delta_{\mathcal{A}_\kappa}(H_{\kappa,\zeta_n}, H'_{\kappa,\zeta_n}) &= \left|\Pr[\mathcal{A}_\kappa(H_{\kappa,\zeta_n}) = 1] - \Pr[\mathcal{A}_\kappa(H'_{\kappa,\zeta_n}) = 1]\right| \\
&= \left|\sum_{\mathbf{s}\in\{0,1\}^n} \Pr[\zeta_n = \mathbf{s}] \left(\Pr[\mathcal{A}_\kappa(H_{\kappa,\mathbf{s}}) = 1] - \Pr[\mathcal{A}_\kappa(H'_{\kappa,\mathbf{s}}) = 1]\right)\right| \\
&\geq \Pr[\zeta_n = 0^n] \left|\Pr[\mathcal{A}_\kappa(H_{\kappa,0^n}) = 1] - \Pr[\mathcal{A}_\kappa(H'_{\kappa,0^n}) = 1]\right| \\
&\quad - \sum_{\mathbf{s}\in\{0,1\}^n\setminus\{0^n\}} \Pr[\zeta_n = \mathbf{s}] \left|\Pr[\mathcal{A}_\kappa(H_{\kappa,\mathbf{s}}) = 1] - \Pr[\mathcal{A}_\kappa(H'_{\kappa,\mathbf{s}}) = 1]\right| \\
&= \Pr[\zeta_n = 0^n]\Delta_{\mathcal{A}_\kappa}(H_{\kappa,0^n}, H'_{\kappa,0^n}) - \sum_{\mathbf{s}\in\{0,1\}^n\setminus\{0^n\}} \Pr[\zeta_n = \mathbf{s}]\Delta_{\mathcal{A}_\kappa}(H_{\kappa,\mathbf{s}}, H'_{\kappa,\mathbf{s}}) \\
&\geq \Pr[\zeta_n = 0^n]\Delta_{\mathcal{A}_\kappa}(H_{\kappa,0^n}, H'_{\kappa,0^n}) - \max_{\mathbf{s}\in\{0,1\}^n\setminus\{0^n\}} \Delta_{\mathcal{A}_\kappa}(H_{\kappa,\mathbf{s}}, H'_{\kappa,\mathbf{s}}),
\end{aligned}$$

where the first inequality is due to the reverse triangle inequality. Note that $\Pr[\zeta_n = 0^n] = 2^{-n+1}$. Therefore,

$$\begin{aligned}
\Delta_{\mathcal{A}_\kappa}(H_{\kappa,0^n}, H'_{\kappa,0^n}) &\leq 2^{n-1} \cdot \left[\Delta_{\mathcal{A}_\kappa}(H_{\kappa,\zeta_n}, H'_{\kappa,\zeta_n}) + \max_{\mathbf{s}\in\{0,1\}^n\setminus\{0^n\}} \Delta_{\mathcal{A}_\kappa}(H_{\kappa,\mathbf{s}}, H'_{\kappa,\mathbf{s}})\right] \\
&\leq 2^{n-1} \cdot \left[\Delta_{\mathcal{A}_\kappa}(H_{\kappa,\zeta_n}, H_{\kappa,\Sigma_n}) + \Delta_{\mathcal{A}_\kappa}(H_{\kappa,\Sigma_n}, H'_{\kappa,\Sigma_n}) + \Delta_{\mathcal{A}_\kappa}(H'_{\kappa,\Sigma_n}, H'_{\kappa,\zeta_n})\right] \\
&\quad + 2^{n-1} \cdot \max_{\mathbf{s}\in\{0,1\}^n\setminus\{0^n\}} \Delta_{\mathcal{A}_\kappa}(H_{\kappa,\mathbf{s}}, H'_{\kappa,\mathbf{s}}) \\
&\leq 2^{n-1}\left[\Delta_{\mathcal{A}_\kappa}(H_{\kappa,\zeta_n}, H_{\kappa,\Sigma_n}) + \Delta_{\mathcal{A}_\kappa}(H'_{\kappa,\Sigma_n}, H'_{\kappa,\zeta_n})\right] + 2^n \cdot \max_{\mathbf{s}\in\{0,1\}^n\setminus\{0^n\}} \Delta_{\mathcal{A}_\kappa}(H_{\kappa,\mathbf{s}}, H'_{\kappa,\mathbf{s}}),
\end{aligned}$$

where the last inequality follows by noting that $\Delta_{\mathcal{A}_\kappa}(H_{\kappa,\Sigma_n}, H'_{\kappa,\Sigma_n}) \leq \max_{\mathbf{s}\in\{0,1\}^n\setminus\{0^n\}} \Delta_{\mathcal{A}_\kappa}(H_{\kappa,\mathbf{s}}, H'_{\kappa,\mathbf{s}})$. This concludes the proof of the claim. □

Since $n = \lfloor \log(\kappa) \rfloor$, it suffices to show that the terms $\Delta_{\mathcal{A}_\kappa}(H_{\kappa,\zeta_n}, H_{\kappa,\Sigma_n})$, $\Delta_{\mathcal{A}_\kappa}(H'_{\kappa,\Sigma_n}, H'_{\kappa,\zeta_n})$ and $\max_{\mathbf{s}\in\{0,1\}^n\setminus\{0^n\}} \Delta_{\mathcal{A}_\kappa}(H_{\kappa,\mathbf{s}}, H'_{\kappa,\mathbf{s}})$ are all negligible, where in the case of computational-WI this should hold with respect to polynomially-bounded adversaries $\mathcal{A}$, and in the case of statistical-WI this should hold for every computationally-unbounded adversary $\mathcal{A}$. We begin with the following claim, that bounds the statistical distance between $H_{\kappa,\mathbf{s}}$ and $H'_{\kappa,\mathbf{s}}$ for every $\mathbf{s}$ of positive Hamming weight.

Claim 8.24. *Let $\mathbf{s} \in \{0,1\}^n$ be a string of Hamming weight at least one. Then,*

$$H_{\kappa,\mathbf{s}} \equiv_{\mathsf{negl}(\kappa)} H'_{\kappa,\mathbf{s}}.$$

Proof. Let us define the sets $I_\mathbf{s} := \{i \in [n] : s_i = 1\}$ and $Z_\mathbf{s} = [n] \setminus I_\mathbf{s}$. By the privacy of the n-out-of-n NPSS and since $Z \neq [n]$, both in $H_{\kappa,\mathbf{s}}$ and $H'_{\kappa,\mathbf{s}}$ the partial assignments $(\mathbf{y}_i)_{i\in Z}$ have the same distribution and let us fix those random variables. Let $J = \{j \in [\ell] : \exists i \in Z,\ \mathbf{y}_i[j] \neq *\}$. Observe that the random variables $(C^y_j, o^y_j)_{j\in J}$ have the same distribution in $H_{\kappa,\mathbf{s}}$ and $H'_{\kappa,\mathbf{s}}$ and we fix them as well. If the commitment scheme is statistically hiding then the remaining commitments $(C^y_j)_{j\in[\ell]\setminus J}$ are statistically-close in $H_{\kappa,\mathbf{s}}$ and $H'_{\kappa,\mathbf{s}}$, and otherwise, in the case of trapdoor commitments, the commitments $(C^y_j)_{j\in[\ell]\setminus J}$ have the same distribution in $H_{\kappa,\mathbf{s}}$ and $H'_{\kappa,\mathbf{s}}$ (since we use the "fake mode"). Fix those commitments as well. Finally, it is not hard to verify that conditioned on the above $(\mathsf{crs}_i, \pi_i)_{i\in[n]}$ have the same distribution in $H_{\kappa,\mathbf{s}}$ and $H'_{\kappa,\mathbf{s}}$. This concludes the proof of the claim. □

The guessing game. It remains to bound the terms $\Delta_{\mathcal{A}_\kappa}(H_{\kappa,\zeta_n}, H_{\kappa,\Sigma_n})$ and $\Delta_{\mathcal{A}_\kappa}(H'_{\kappa,\Sigma_n}, H'_{\kappa,\zeta_n})$. To do so, we show that every adversary $\mathcal{A}$ that distinguishes H_{κ,ζ_n} from H_{κ,Σ_n} with non-negligible advantage, can be transformed into a guesser in the guessing game of the algorithms Prove and Sim with non-negligible advantage (see Section 8.1.2 for the definition of the guessing game).

Formally, for a polynomial $p(\kappa)$ we define REAL^p_κ to be the algorithm that (1) takes as an input a circuit F and an assignment $\mathbf{m}$, (2) verifies that $|F|, |\mathbf{m}| \leq p(\kappa)$, and (3) if the verification fails outputs $\perp$ and otherwise samples $(\mathsf{crs}, \pi) \leftarrow \mathsf{Prove}_\kappa(F, \mathbf{m})$ and outputs (crs, π). Similarly, we define IDEAL^p_κ to be the algorithm that (1) takes as an input a circuit F and an assignment $\mathbf{m}$, (2) verifies that $|F|, |\mathbf{m}| \leq p(\kappa)$, and (3) if the verification fails outputs $\perp$ and otherwise samples $(\mathsf{crs}, \pi) \leftarrow \mathsf{Sim}_{wk}(1^\kappa, F)$ and outputs (crs, π). In the following, we let $p_f(\kappa)$ be a polynomial that satisfies $\left|\hat{f}_1\right|, \ldots, \left|\hat{f}_n\right| \leq p_f(\kappa)$, where $\hat{f}_1, \ldots, \hat{f}_n$ are defined as in $\mathcal{C}_{0^n}(1^\kappa, f_\kappa, \mathbf{x}^0_\kappa)$ or $\mathcal{C}_{0^n}(1^\kappa, f_\kappa, \mathbf{x}^1_\kappa)$.

Claim 8.25. *If Π_{wk} satisfies $1/\kappa$-computational-WI then, for for every polynomial s and all sufficiently large κ it holds that $\mathsf{Guess}^s(\text{REAL}^{p_f}_\kappa, \mathsf{Ideal}^{p_f}_\kappa, 1) \leq 1/\kappa$. Furthermore, if Π_{wk} satisfies $1/\kappa$-statistical-WI, then for all sufficiently large κ it holds that $\mathsf{Guess}(\text{REAL}^{p_f}_\kappa, \mathsf{Ideal}^{p_f}_\kappa, 1) \leq 1/\kappa$.*

Proof. Assume, towards a contradiction, that the first part of the claim does not hold. Then there is a polynomially-bounded adversary $\mathcal{A} = \{\mathcal{A}_\kappa\}_{\kappa\in\mathbb{N}}$ that satisfies $\mathsf{Guess}^{\mathcal{A}_\kappa}(\text{REAL}^{p_f}_\kappa, \text{IDEAL}^{p_f}_\kappa, 1) > 1/\kappa$ for infinitely many κ's. We may assume without loss of generality that $\mathcal{A}$ is deterministic, and that the single query of $\mathcal{A}_\kappa$ is $(F_\kappa, \mathbf{m}_\kappa)$ with $|F_\kappa|, |\mathbf{m}_\kappa| \leq p_f(\kappa)$. This violates the $(1/\kappa)$-WI with respect to the ensemble $\{F_\kappa, \mathbf{m}_\kappa\}_{\kappa\in\mathbb{N}}$, and we derive a contradiction. If the adversary is computationally-unbounded, this violates $(1/\kappa)$-statistical-WI, and the second part of the claim follows. □

Claim 8.26. *Let $\mathcal{A} = \{\mathcal{A}_\kappa\}_{\kappa\in\mathbb{N}}$ be a distinguisher, and for every $\kappa \in \mathbb{N}$ denote $\delta_\mathcal{A}(\kappa) := \Delta_{\mathcal{A}_\kappa}(H_{\kappa,\zeta_n}, H_{\kappa,\Sigma_n})$ (resp., $\delta_\mathcal{A}(\kappa) := \Delta_{\mathcal{A}_\kappa}(H'_{\kappa,\zeta_n}, H'_{\kappa,\Sigma_n})$). Then there is a distinguisher $\mathcal{B} = \{\mathcal{B}_\kappa\}_{\kappa\in\mathbb{N}}$ of complexity polynomial in that of $\mathcal{A}$, such that $\mathsf{Guess}^{\mathcal{B}_\kappa}(\text{REAL}^{p_f}_\kappa, \text{IDEAL}^{p_f}_\kappa, n(\kappa)) = \delta_\mathcal{A}(\kappa)$. In addition, the queries of $\mathcal{B}_\kappa$ are of the form $(F, \mathbf{m})$ with $|F|, |\mathbf{m}| \leq p_f(\kappa)$.*

Proof. We focus on the case where $\delta_\mathcal{A}(\kappa) := \Delta_{\mathcal{A}_\kappa}(H_{\kappa,\zeta_n}, H_{\kappa,\Sigma_n})$. The case where $\delta_\mathcal{A}(\kappa) := \Delta_{\mathcal{A}_\kappa}(H'_{\kappa,\zeta_n}, H'_{\kappa,\Sigma_n})$ follows in a similar way by replacing $\mathbf{x}^0_\kappa$ with $\mathbf{x}^1_\kappa$. Let us assume without loss of generality that

$$\delta_\mathcal{A}(\kappa) = \Pr[\mathcal{A}_\kappa(H_{\kappa,\Sigma_n}) = 1] - \Pr[\mathcal{A}_\kappa(H_{\kappa,\zeta_n}) = 1] \geq 0.$$

The adversary $\mathcal{B}$. Let $\mathcal{B}_\kappa$ be the adversary that samples $\mathbf{C}^y, (\hat{f}_i, \hat{\mathbf{y}}_i)_{i\in[n]}$ like in $\mathcal{C}_{0^n}(1^\kappa, f_\kappa, \mathbf{x}^0_\kappa)$ (observe that the distribution of $\mathbf{C}^y, (\hat{f}_i, \hat{\mathbf{y}}_i)_{i\in[n]}$ in $\mathcal{C}_\mathbf{s}(1^\kappa, f_\kappa, \mathbf{x}^0_\kappa)$ is the same for every choice of $\mathbf{s}$, and for concreteness we take $\mathbf{s} = 0^n$). Then for every $i \in [n]$, $\mathcal{B}$ calls the oracle F_i with the query $(\hat{f}_i, \hat{\mathbf{y}}_i)$ to obtain (crs_i, π_i). The adversary $\mathcal{B}_\kappa$ runs $\mathcal{A}_\kappa(\mathsf{pp}, (\mathsf{crs}_i)_{i\in[n]}, \mathbf{C}^y, (\pi_i)_{i\in[n]}) \to b$ and outputs b. Observe that if the oracles are chosen according to a string $\mathbf{s}^* \in \{0,1\}^n$, then $\mathbf{C}^y, (\mathsf{crs}_i, \pi_i)_{i\in[n]}$ have the same distribution as in $\mathcal{C}_{\mathbf{s}^*}(1^\kappa, f_\kappa, \mathbf{x}^0_\kappa)$.

Analysis. It holds that

$$\begin{aligned}
\Pr_{\mathbf{s}\leftarrow\{0,1\}^n}\left[\mathcal{B}^{F_1,\ldots,F_n} = \oplus_{i\in[n]} s_i\right] &= \Pr_{\mathbf{s}}[b = \oplus_{i\in[n]} s_i] \\
&= \frac{1}{2}\left[\Pr_{\mathbf{s}}[b = 0 \mid \oplus_{i\in[n]} s_i = 0] + \Pr_{\mathbf{s}}[b = 1 \mid \oplus_{i\in[n]} s_i = 1]\right] \\
&= \frac{1}{2}\left[\Pr_{\mathbf{s}}[\mathcal{A}_\kappa(H_{\kappa,\zeta_n}) = 0] + \Pr_{\mathbf{s}}[\mathcal{A}_\kappa(H_{\kappa,\Sigma_n}) = 1]\right] \\
&= \frac{1}{2}\left[1 + \Pr_{\mathbf{s}}[\mathcal{A}_\kappa(H_{\kappa,\Sigma_n}) = 1] - \Pr_{\mathbf{s}}[\mathcal{A}_\kappa(H_{\kappa,\zeta_n}) = 1]\right] \\
&= \frac{1}{2} + \frac{\delta_{\mathcal{A}}(\kappa)}{2}.
\end{aligned}$$

This concludes the proof of the claim. □

Completing the proof. We continue with the case of statistical-WI and statistically-hiding commitments (resp., computational-WI). Fix any non-uniform computationally-unbounded (resp,. computationally-bounded) distinguisher $\mathcal{A} = \{\mathcal{A}_\kappa\}_{\kappa\in\mathbb{N}}$ of size $\sigma(\kappa)$. Recall that $n = \lfloor \log(n) \rfloor$ and therefore, by Claim 8.23, it suffices to show that the terms $\Delta_{\mathcal{A}_\kappa}(H_{\kappa,\zeta_n}, H_{\kappa,\Sigma_n}), \Delta_{\mathcal{A}_\kappa}(H'_{\kappa,\zeta_n}, H'_{\kappa,\Sigma_n})$ and $\max_{\mathbf{s}\in\{0,1\}^n\setminus\{0^n\}} \Delta_{\mathcal{A}_\kappa}(H_{\kappa,\mathbf{s}}, H'_{\kappa,\mathbf{s}})$ are bounded by a negligible function. For the last term, this follows from Claim 8.24, and we therefore focus on the first two terms. By Claim 8.26, if one of the first terms is not bounded by a negligible function, then there is a distinguisher $\mathcal{B} = \{\mathcal{B}_\kappa\}_{\kappa\in\mathbb{N}}$ of size $\sigma'(\kappa) = \mathrm{poly}(\sigma(\kappa))$ and a polynomial q such that $\mathsf{Guess}^{\mathcal{B}_\kappa}(\mathrm{REAL}_\kappa^{p_f}, \mathrm{IDEAL}_\kappa^{p_f}, n(\kappa)) > 1/q(\kappa)$ for infinitely many κ's. We split into cases.

- **Statistical-WI and statistically-hiding commitments.** Theorem 8.4 (the information-theoretic XOR lemma) implies that for all sufficiently large κ it holds that $\mathsf{Guess}(\mathrm{REAL}_\kappa^{p_f}, \mathrm{IDEAL}_\kappa^{p_f}, n(\kappa)) \leq \mathsf{Guess}(\mathrm{REAL}_\kappa^{p_f}, \mathrm{IDEAL}_\kappa^{p_f}, 1)^{n(\kappa)} \leq 1/\kappa^n = \mathsf{negl}(\kappa)$, where in the last inequality we used Claim 8.25. We therefore obtain that for infinitely many κ's it holds that $1/q(\kappa) < \mathsf{negl}(\kappa)$, in contradiction.

- **Computational-WI.** Next, let us consider the case of computational-WI. Claim 8.25 implies that for every polynomial s there exists κ_0 such that for all $\kappa \geq \kappa_0$ it holds that $\mathsf{Guess}^{s(\kappa)}(\mathrm{REAL}_\kappa^{p_f}, \mathrm{IDEAL}_\kappa^{p_f}, 1) \leq 1/\kappa$. Then Theorem 8.5, when applied on $n(\kappa)$, $\epsilon(\kappa) = 1/(2q(\kappa)\cdot e^{n(\kappa)-1})$ and a sufficiently large polynomial s, implies that for all sufficiently large κ it holds that

  $$\mathsf{Guess}^{\sigma'(\kappa)}(\mathrm{REAL}_\kappa^{p_f}, \mathrm{IDEAL}_\kappa^{p_f}, n(\kappa)) \leq e^{n(\kappa)-1}\left(\frac{1}{\kappa^{n(\kappa)}} + \epsilon(\kappa)\right) = \mathsf{negl}(\kappa) + \frac{1}{2q(\kappa)},$$

  where we used the fact that $n = \lfloor \log(\kappa) \rfloor$. We therefore obtain that for infinitely many κ's it holds that $1/q(\kappa) < \mathsf{negl}(\kappa) + 1/(2q(\kappa))$, in contradiction.

This concludes the proof of the lemma. □

8.4 Putting it All Together

We prove the following theorem.

Theorem 8.27 (Theorem 1.1 restated.)**.** *Let Π_{wk} be an (ϵ_s, ϵ_z)-weak NIZK, where*

$$\epsilon_s(\kappa) + \epsilon_z(\kappa) < 1 - \frac{1}{\kappa^c},$$

for some constant c. Then,

1. *If Π_{wk} satisfies ϵ_s-statistical-soundness and ϵ_z-computational-zero-knowledge, then assuming the existence of one-way functions there exists a NIZK Π with statistical soundness and computational zero knowledge.*

2. *If Π_{wk} satisfies ϵ_s-computational-soundness and ϵ_z-computational-zero-knowledge, then assuming the existence of public-key encryption there exists a NIZK Π with computational soundness and zero knowledge.*

3. *If Π_{wk} satisfies ϵ_s-proof-of-knowledge and ϵ_z-statistical-zero-knowledge, then assuming the existence of collision-resistant hash functions there exists a NIZK Π with proof of knowledge and statistical zero knowledge.*

Proof. At a high level, we first use the amplifier from Figure 18 to amplify statistical-soundness/proof-of-knowledge to negligible and WI to inverse polynomial, and then we use the amplifier from Figure 20 to reduce the WI error to negligible. If we begin with computational soundness (Item 2 of the theorem), then we first use Lemma 8.13 to obtain a scheme with proof of knowledge. This transformation inflicts a negligible loss in the parameters, i.e., we obtain $\epsilon'_p(\kappa) = \epsilon_s(\kappa) + \mathsf{negl}(\kappa)$ proof of knowledge, and $\epsilon'_z(\kappa) := \epsilon_z(\kappa) + \mathsf{negl}(\kappa)$ zero knowledge, and therefore we are only guaranteed that $\epsilon'_p(\kappa) + \epsilon'_z(\kappa) < 1 - \frac{1}{\kappa^c} + \mathsf{negl}(\kappa) \leq 1 - \frac{1}{\kappa^{2c}}$. Therefore, it will be convenient to pick parameters for all the cases with respect to the bound $1 - \frac{1}{\kappa^{2c}}$. Let us set

$$n(\kappa) = 70\kappa^{6c} \cdot (\kappa + \ln(125\kappa^{4c})), \quad t(\kappa) = \left\lceil \epsilon_z(\kappa) + \frac{1}{10\kappa^{2c}} \right\rceil n, \quad \text{and} \quad \alpha(\kappa) = 100\kappa^{2c+3}.$$

Observe that $2 \leq t \leq n$, and in fact $t(\kappa) \leq (1 - \epsilon_s(\kappa) - \frac{4}{5\kappa^{2c}})$. In case of computational zero knowledge (Items 1 and 2) we let $\Pi_{\mathsf{TCom}} = (\mathsf{Gen}_{\mathsf{Commit}}, \mathsf{Gen}'_{\mathsf{Commit}}, \mathsf{Commit}, \mathsf{VOpen}, \mathsf{TCom})$ be the trapdoor commitment scheme promised in Theorem 8.3, and we set $\Pi_{\mathsf{Commit}} = (\mathsf{Gen}_{\mathsf{Commit}}, \mathsf{Commit}, \mathsf{VOpen})$ and recall that Π_{Commit} is a statistically-binding non-interactive commitment scheme. In the case of statistical zero knowledge (Item 3) we let Π_{Commit} be the statistically-hiding commitment scheme that is based on collision-resistant hash functions (see Section 8.1.1). Recall that WI is defined with respect to an inefficient simulator (see Definition A.9), and therefore ϵ_z-WI (resp., ϵ_z-statistical-WI) is implied by ϵ_z-zero-knowledge (resp., ϵ_z-statistical-zero-knowledge). We split into cases.

1. Assume that Π_{wk} satisfies ϵ_s-statistical-soundness and ϵ_z-computational-zero-knowledge. By applying the amplifier $\mathsf{Amplifier}_{\alpha,t,n,\Pi_{\mathsf{Commit}}}$ from Figure 18 on Π_{wk} we obtain a scheme Π' that (a) by Lemma 8.16 satisfies statistical soundness, and (b) by Lemma 8.14, when applied with $\delta_w(\kappa) = 1/\kappa$, satisfies $1/\kappa$-computational-WI. We then apply the amplifier $\mathsf{Amplifier}^{WI}_{\lfloor \log(\kappa) \rfloor, \Pi_{\mathsf{Commit}}}$ from Figure 20 on Π' to obtain a scheme Π'' that (a) by Lemma 8.22 satisfies statistical soundness, and (b) by Lemma 8.21 satisfies WI. Finally, we apply Lemma 8.11 to obtain a NIZK that satisfies statistical soundness and zero knowledge.

2. Assume that Π_{wk} satisfies ϵ_s-computational-soundness and ϵ_z-computational-zero-knowledge. In this case we first apply Lemma 8.13 that relies on PKE to obtain a scheme Π' with $(\epsilon_s + \mathsf{negl})$-proof of knowledge and $(\epsilon_z + \mathsf{negl})$-computational-zero-knowledge. By applying the amplifier $\mathsf{Amplifier}_{\alpha,t,n,\Pi_{\mathsf{Commit}}}$ from Figure 18 on Π' we obtain a scheme Π'' that (a) by Lemma 8.15 satisfies proof of knowledge, and (b) by Lemma 8.14, when applied with $\delta_w(\kappa) = 1/\kappa$, satisfies $1/\kappa$-computational-WI. We then apply the amplifier $\mathsf{Amplifier}^{WI}_{\lfloor \log(\kappa) \rfloor,\Pi_{\mathsf{Commit}}}$ from Figure 20 on Π' to obtain a scheme Π'' that (a) by Lemma 8.22 satisfies proof of knowledge, and (b) by Lemma 8.21 satisfies WI. Finally, we apply Lemma 8.11 to obtain a NIZK that satisfies soundness and zero knowledge.

3. Assume that Π_{wk} satisfies ϵ_s-proof-of-knowledge and ϵ_z-statistical-zero-knowledge. By applying the amplifier $\mathsf{Amplifier}_{\alpha,t,n,\Pi_{\mathsf{Commit}}}$ from Figure 18 on Π_{wk} we obtain a scheme Π' that (a) by Lemma 8.15 satisfies proof of knowledge, and (b) by Lemma 8.14, when applied with $\delta_w(\kappa) = 1/\kappa$, satisfies $1/\kappa$-statistical-WI. By applying the amplifier $\mathsf{Amplifier}^{WI}_{\lfloor \log(\kappa) \rfloor,\Pi_{\mathsf{Commit}}}$ from Figure 20 on Π' we obtain a scheme Π'' that by Lemma 8.22 satisfies proof of knowledge, and by Lemma 8.21 statistical-WI. Finally, we apply Lemma 8.12 to obtain a scheme with proof of knowledge and statistical soundness.

This concludes the proof of the theorem.

□

Remark 8.28 (Preserving succinctness for statistical zero knowledge.)**.** *When the original scheme Π_{wk} is* succinct*, i.e., its proof length for S-size circuits is $\mathrm{poly}(\kappa, \log(S))$, we can slightly modify the amplifier to preserve succinctess in the case of ϵ_s-proof-of-knowledge and ϵ_z-statistical-zero-knowledge. This is done by replacing Π_{Commit} with statistically-hiding* succinct commitments*, i.e., commitments whose length is $\mathrm{poly}(\kappa, \log(\ell))$ when committing on ℓ bits. We also require Π_{Commit} to have* local openings*, i.e., the opening/decommitment information o generated by $\mathsf{Commit}_{\mathsf{pp}}(\mathbf{x})$ for $\mathbf{x} \in \{0,1\}^\ell$ is composed of ℓ local commitments $(o_i)_{i\in[\ell]}$ each of length $\mathrm{poly}(\kappa) \cdot \log(\ell)$. Correctness should hold separately for each bit, i.e., $\mathsf{VOpen}_{\mathsf{pp}}(C_i, o_i, \mathbf{x}[i]) = 1$, and the opening algorithm should run in time $\mathrm{poly}(\kappa) \cdot \log(\ell)$. Hiding should hold even when some of the bits were "opened". We note that such commitments can be based on collision-resistant hash functions by using a Merkle tree on the construction of [DPP98, HM96]. Now, in $\mathsf{Amplifier}_{\alpha,t,n,\Pi_{\mathsf{Commit}}}$ and $\mathsf{Amplifier}^{WI}_{\lfloor \log(\kappa) \rfloor,\Pi_{\mathsf{Commit}}}$, the prover commits to the global assignment $\mathbf{y}$ using the succinct commitments with local openings. In this way, the total length of the commitment is $\mathrm{poly}(\kappa, \log(|f|))$ and the size of each circuit $\hat{f}_i$ is $\mathrm{poly}(\kappa, |f|)$ so the length of π_i is $\mathrm{poly}(\kappa, \log(|f|))$, and therefore the new proof is succinct.*

Acknowledgment. We thank Nathan Geier for helpful discussions and Nir Bitansky for suggesting the use of trapdoor commitments.

References

[AK25] Benny Applebaum and Eliran Kachlon. How to share an NP statement or combiners for zero-knowledge proofs. Cryptology ePrint Archive, Paper 2025/334, 2025. To appear in Crypto'25.

[AKS83] Miklós Ajtai, János Komlós, and Endre Szemerédi. An $O(n \log n)$ sorting network. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 1–9, 1983.

[App17] Benny Applebaum. Garbled circuits as randomized encodings of functions: a primer. In *Tutorials on the Foundations of Cryptography.*, pages 1–44. 2017.

[BDIR21] Fabrice Benhamouda, Akshay Degwekar, Yuval Ishai, and Tal Rabin. On the local leakage resilience of linear secret sharing schemes. *Journal of Cryptology*, 34:1–65, 2021.

[BDSL14] Nir Bitansky, Dana Dachman-Soled, and Huijia Lin. Leakage-tolerant computation with input-independent preprocessing. In *Advances in Cryptology–CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II 34*, pages 146–163. Springer, 2014.

[Bel02] Mihir Bellare. A note on negligible functions. *J. Cryptol.*, 15(4):271–284, 2002.

[BG24] Nir Bitansky and Nathan Geier. Amplification of non-interactive zero knowledge, revisited. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part IX*, volume 14928 of *Lecture Notes in Computer Science*, pages 361–390. Springer, 2024.

[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*, pages 1235–1254, 2012.

[BGK14] Elette Boyle, Shafi Goldwasser, and Yael Tauman Kalai. Leakage-resilient coin tossing. *Distributed computing*, 27:147–164, 2014.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10. ACM, 1988.

[BKP+24] Nir Bitansky, Chethan Kamath, Omer Paneth, Ron D Rothblum, and Prashant Nalini Vasudevan. Batch proofs are statistically hiding. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 435–443, 2024.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 11–19. ACM, 1988.

[CDI+13] Gil Cohen, Ivan Bjerre Damgård, Yuval Ishai, Jonas Kölker, Peter Bro Miltersen, Ran Raz, and Ron D. Rothblum. Efficient multiparty protocols via log-depth threshold formulae - (extended abstract). In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 185–202. Springer, 2013.

[DKS99] Ivan Damgård, Joe Kilian, and Louis Salvail. On the (im) possibility of basing oblivious transfer and bit commitment on weakened security assumptions. In *Advances in Cryptology—EUROCRYPT'99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings 18*, pages 56–73. Springer, 1999.

[DNR04] Cynthia Dwork, Moni Naor, and Omer Reingold. Immunizing encryption schemes from decryption errors. In *Advances in Cryptology-EUROCRYPT 2004: International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004. Proceedings 23*, pages 342–360. Springer, 2004.

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

[DPP98] Ivan Damgård, Torben P. Pedersen, and Birgit Pfitzmann. Statistical secrecy and multibit commitments. *IEEE Trans. Inf. Theory*, 44(3):1143–1151, 1998.

[FGMO05] Matthias Fitzi, Juan A. Garay, Ueli M. Maurer, and Rafail Ostrovsky. Minimal complete primitives for secure multi-party computation. *J. Cryptol.*, 18(1):37–61, 2005.

[FKN94] Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). pages 554–563, 1994.

[FLS90] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string. In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 308–317. IEEE, 1990.

[FS89] Uriel Feige and Adi Shamir. Zero knowledge proofs of knowledge in two rounds. In *Conference on the Theory and Application of Cryptology*, pages 526–544. Springer, 1989.

[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. IEEE, 2016.

[GJS19] Vipul Goyal, Aayush Jain, and Amit Sahai. Simultaneous amplification: The case of non-interactive zero-knowledge. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 608–637. Springer, 2019. Crypto talk avaialable in https://www.youtube.com/watch?v=DLK079q6aWM&ab_channel=IACR.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229, 1987.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *Journal of the ACM (JACM)*, 38(3):690–728, 1991.

[GNW11] Oded Goldreich, Noam Nisan, and Avi Wigderson. On yao's xor-lemma. *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation: In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman*, pages 273–301, 2011.

[Gol01] Oded Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001.

[GP99] Oded Goldreich and Erez Petrank. Quantifying knowledge complexity. *Comput. Complex.*, 8(1):50–98, 1999.

[GR15] Shafi Goldwasser and Guy N Rothblum. How to compute in the presence of leakage. *SIAM Journal on Computing*, 44(5):1480–1549, 2015.

[HM96] Shai Halevi and Silvio Micali. Practical and provably-secure commitment schemes from collision-free hashing. pages 201–215, 1996.

[HM00] Martin Hirt and Ueli M. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *J. Cryptol.*, 13(1):31–60, 2000.

[Hol05] Thomas Holenstein. Key agreement from weak bit agreement. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 664–673, 2005.

[HR08] Shai Halevi and Tal Rabin. Degradation and amplification of computational hardness. In *Theory of Cryptography: Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008. Proceedings 5*, pages 626–643. Springer, 2008.

[IJK09] Russell Impagliazzo, Ragesh Jaiswal, and Valentine Kabanets. Chernoff-type direct product theorems. *Journal of Cryptology*, 22(1):75–92, 2009.

[IK02] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. pages 244–256, 2002.

[Imp95] Russell Impagliazzo. Hard-core distributions for somewhat hard problems. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*, pages 538–545. IEEE Computer Society, 1995.

[Ish13] Yuval Ishai. Randomization techniques for secure computation. In *Secure Multi-Party Computation*, pages 222–248. IOS press, 2013.

[Jai24] Aayush Jain. Aayush jain's website, 2024. https://sites.google.com/view/aayushjain/home.

[LM20] David Lanzenberger and Ueli Maurer. Coupling of random systems. In *Theory of Cryptography Conference*, pages 207–240. Springer, 2020.

[LT13] Huijia Lin and Stefano Tessaro. Amplification of chosen-ciphertext security. In *Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32*, pages 503–519. Springer, 2013.

[MT10] Ueli M. Maurer and Stefano Tessaro. A hardcore lemma for computational indistinguishability: Security amplification for arbitrarily weak prgs with optimal stretch. In Daniele Micciancio, editor, *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*, volume 5978 of *Lecture Notes in Computer Science*, pages 237–254. Springer, 2010.

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

[Pat90] Michael S Paterson. Improved sorting networks with o (log n) depth. *Algorithmica*, 5(1):75–92, 1990.

[Sei09] Joel Seiferas. Sorting networks of logarithmic depth, further simplified. *Algorithmica*, 53:374–384, 2009.

[Spi71] Philip Spira. On time-hardware complexity tradeoffs for boolean functions. In *Proceedings of the 4th Hawaii Symposium on System Sciences, 1971*, pages 525–527, 1971.

[SV19] Akshayaram Srinivasan and Prashant Nalini Vasudevan. Leakage resilient secret sharing and applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 480–509. Springer, 2019.

[Tes10] Stefano Tessaro. *Computational Indistinguishability Amplification*. PhD thesis, ETH Zurich, Zürich, Switzerland, 2010.

[Weg83] Ingo Wegener. Relating monotone formula size and monotone depth of boolean functions. *Inf. Process. Lett.*, 16(1):41–42, 1983.

[Wul07] Jürg Wullschleger. Oblivious-transfer amplification. In *Advances in Cryptology-EUROCRYPT 2007: 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007. Proceedings 26*, pages 555–572. Springer, 2007.

[Yao82] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). pages 80–91, 1982.

A Preliminaries

A.1 Statistical Distance and Entropy

Basic notation. Let A and B be two random variables. If A and B have the same distribution then we denote $A \equiv B$. If A the statistical distance between A and B is at most ϵ then we denote $A \equiv_\epsilon B$ or $\mathsf{SD}(A, B) \leq \epsilon$, and we say that A is ϵ-close to B. If the statistical distance between A and B is larger than ϵ then we denote $\mathsf{SD}(A, B) > \epsilon$ and we say that A is ϵ-far from B.

Conditional min-entropy. The following is taken, with some modifications, from [DORS08]. For a random variable A the *min-entropy* of A, denoted $\mathsf{H}_\infty(A)$ is defined to be $-\log(\max_a \Pr[A = a])$. For a pair of (possibly correlated) random variables A, B, the *average min-entropy* of A given B is given by

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

Definition A.1. *A function* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^\ell$ *is an* average-case (m, ϵ)-strong extractor *if for all pairs of random variables* (W, I) *such that* W *is an* n*-bit string satisfying* $\tilde{\mathsf{H}}_\infty(W \mid I) \geq m$*, we have* $\mathsf{SD}((\mathsf{Ext}(W; X), X, I), (U_\ell, X, I)) \leq \epsilon$*, where* X *is uniform over* $\{0,1\}^d$*.*

Definition A.2. *Let* $\mathcal{H}$ *be a family of functions* $h : \{0,1\}^n \to \{0,1\}^\ell$*. Then* $\mathcal{H}$ *is* universal *if for all* $a \neq b \in \{0,1\}^n$ *it holds that* $\Pr_{h \leftarrow \mathcal{H}}[h(a) = h(b)] = 2^{-\ell}$*.*

Lemma A.3. *Let* $\mathcal{H}$ *be a universal family of functions* $h : \{0,1\}^n \to \{0,1\}^\ell$*. Then for any random variables* W *and* I*,*

$$\mathsf{SD}((h(W), h, I), (U_\ell, h, I)) \leq \frac{1}{2}\sqrt{2^{-\tilde{\mathsf{H}}_\infty(W|I)} \cdot 2^\ell},$$

where $h \leftarrow \mathcal{H}$*. In particular, if a function from* $\mathcal{H}$ *can be sampled given* d *random bits, then the function* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^\ell$ *that takes a seed* $W \in \{0,1\}^n$ *and a description of a function* $h \in \mathcal{H}$ *and outputs* $h(W)$*, is an average-case* (m, ϵ)*-strong extractor whenever* $\ell \leq m - 2\log(\frac{1}{\epsilon}) + 2$*.*

Since the set of all degree-1 univariate polynomials over $\mathbb{F}_{2^n}$ is pairwise independent (and therefore also universal), we obtain the following corollary.

Corollary A.4. *For every* $m \leq n$*,* $\epsilon > 0$ *and* $\ell \leq m - 2\log(\frac{1}{\epsilon}) + 2$ *there exists an explicit construction of an an average-case* (m, ϵ)*-strong extractor* $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^{2n} \to \{0,1\}^\ell$*.*

A.2 Cryptographic Primitives

For every pair of ensembles $X = \{X_\kappa\}_{\kappa \in \mathbb{N}}$ and $Y = \{Y_\kappa\}_{\kappa \in \mathbb{N}}$ and a function $\epsilon(\kappa)$ we write $X \approx_{\epsilon(\kappa)} Y$ against $s(n)$-sized circuits if for every $s(n)$-sized non-uniform family of circuits $\mathcal{D} = \{\mathcal{D}_\kappa\}_{\kappa \in \mathbb{N}}$ and large enough $\kappa \in \mathbb{N}$ it holds that

$$|\Pr[\mathcal{D}_\kappa(X_\kappa) = 1] - \Pr[\mathcal{D}_\kappa(Y_\kappa) = 1]| \leq \epsilon(\kappa).$$

We omit the bound on the size when we consider polynomial-sized family of circuits, and we omit ϵ when it is a negligible function. We say that X is computationally-indistinguishable from Y, if for every polynomial-sized non-uniform family of circuits $\mathcal{D} = \{\mathcal{D}_\kappa\}_{\kappa \in \mathbb{N}}$ there is a negligible function μ, so that for all sufficiently large $\kappa \in \mathbb{N}$ it holds that

$$|\Pr[\mathcal{D}_\kappa(X_\kappa) = 1] - \Pr[\mathcal{D}_\kappa(Y_\kappa) = 1]| \leq \mu(\kappa).$$

A.2.1 Non-Interactive Zero-Knowledge

The following is taken, with some changes, from [BG24]. The definitions are given with respect to circuit-SAT.

Definition A.5 (Non-interactive protocol in the CRS model)**.** *A* non-interactive protocol *(NIP)* Π *in the CRS model for circuit-SAT is a tuple of PPT algorithms* $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$*, with the following syntax:*

- $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\kappa)$*: Given the security parameter* κ*, the randomized set-up algorithm* Gen *outputs a CRS* crs*.*
- $\pi \leftarrow \mathcal{P}(1^\kappa, \mathsf{crs}, f, \mathbf{x})$*: Given the security parameter* κ*, a CRS* crs*, a circuit-SAT instance* f *and an assignent* $\mathbf{x}$ *the randomized prover outputs a proof* π*.*
- $b = \mathcal{V}(1^\kappa, \mathsf{crs}, f, \pi)$*: Given the security parameter* κ*, a CRS* crs*, a circuit-SAT instance* f*, and a proof* π*, the deterministic verifier returns a bit* b*, representing accept or reject.*

We continue with the definitions of completeness, soundness, zero-knowledge, and witness indistinguishability of a NIP Π.

Definition A.6 (Completeness)**.** *An NIP* Π *satisfies* completeness *if for every ensemble* $\{(f_\kappa, \mathbf{x}_\kappa)\}_{\kappa\in\mathbb{N}}$ *and polynomial* p *that satisfy* $f_\kappa(\mathbf{x}_\kappa) = 1$ *and* $|f_\kappa|, |\mathbf{x}_\kappa| \leq p(\kappa)$ *for all* $\kappa \in \mathbb{N}$*, there exists a negligible function* μ *so that for all* $\kappa \in \mathbb{N}$*,*

$$\Pr_{\substack{\mathsf{crs}\leftarrow\mathsf{Gen}(1^\kappa)\\ \pi\leftarrow\mathcal{P}(1^\kappa,\mathsf{crs},f_\kappa,\mathbf{x}_\kappa)}}[\mathcal{V}(1^\kappa, \mathsf{crs}, f_\kappa, \pi) = 1] > 1 - \mu(\kappa).$$

Definition A.7 (Adaptive soundness)**.** *An NIP* Π *is* $\epsilon_s(\kappa)$-computationally-sound *if for every polynomial-sized non-uniform circuit family of malicious provers* $\mathcal{P}^* = \{\mathcal{P}^*_\kappa\}_{\kappa\in\mathbb{N}}$ *and all sufficiently large* κ*,*

$$\Pr_{\substack{\mathsf{crs}\leftarrow\mathsf{Gen}(1^\kappa)\\ (f,\pi)\leftarrow\mathcal{P}^*_\kappa(1^\kappa,\mathsf{crs})}}[f \text{ is not satisfiable} \wedge \mathcal{V}(1^\kappa, \mathsf{crs}, f, \pi) = 1] < \epsilon_s(\kappa).$$

If the above holds also for unbounded circuit families $\mathcal{P}^*$ *with polynomially-bounded output-length, we say that* Π *is* $\epsilon_s(\kappa)$*-statistically-sound. We may omit* ϵ_s *when it is negligible.*

Definition A.8 (Zero knowledge)**.** *An NIP* Π *is* $\epsilon_z(\kappa)$-zero knowledge *if there exists a PPT simulator* Sim *so that for every ensemble* $\{(f_\kappa, \mathbf{x}_\kappa)\}_{\kappa\in\mathbb{N}}$ *and polynomial* p *that satisfy* $f_\kappa(\mathbf{x}_\kappa) = 1$ *and* $|f_\kappa|, |\mathbf{x}_\kappa| \leq p(\kappa)$ *for every* $\kappa \in \mathbb{N}$*, for every polynomial-sized non-uniform circuit family* $\mathcal{D} = \{\mathcal{D}_\kappa\}_{\kappa\in\mathbb{N}}$*, and for all sufficiently large* κ*,*

$$\left|\Pr_{\substack{\mathsf{crs}\leftarrow\mathsf{Gen}(1^\kappa)\\ \pi\leftarrow\mathcal{P}(1^\kappa,\mathsf{crs},f_\kappa,\mathbf{x}_\kappa)}}[\mathcal{D}_\kappa(1^\kappa, \mathsf{crs}, \pi) = 1] - \Pr_{(\mathsf{crs},\pi)\leftarrow\mathsf{Sim}(1^\kappa,f_\kappa)}[\mathcal{D}_\kappa(1^\kappa, \mathsf{crs}, \pi) = 1]\right| < \epsilon_z(\kappa).$$

If the above holds also for unbounded circuit families $\mathcal{D}$ *then we say that* Π *is* $\epsilon_z(\kappa)$*-statistically-zero-knowledge.*

Definition A.9 (Witness indistinguishability)**.** *An NIP* Π *satisfies* $\epsilon_w(\kappa)$-witness indistinguishability *(WI) if there exists a (possibly inefficient) simulator* Sim *so that for every ensemble* $\{(f_\kappa, \mathbf{x}_\kappa)\}_{\kappa\in\mathbb{N}}$ *and polynomial* p *that satisfy* $f_\kappa(\mathbf{x}_\kappa) = 1$ *and* $|f_\kappa|, |\mathbf{x}_\kappa| \leq p(\kappa)$ *for every* $\kappa \in \mathbb{N}$*, for every polynomial-sized non-uniform circuit family* $\mathcal{D} = \{\mathcal{D}_\kappa\}_{\kappa\in\mathbb{N}}$*, and for all sufficiently large* κ*,*

$$\left| \Pr_{\substack{\mathsf{crs}\leftarrow\mathsf{Gen}(1^\kappa)\\ \pi\leftarrow\mathcal{P}(1^\kappa,\mathsf{crs},f_\kappa,\mathbf{x}_\kappa)}} [\mathcal{D}_\kappa(1^\kappa,\mathsf{crs},\pi)=1] - \Pr_{(\mathsf{crs},\pi)\leftarrow\mathsf{Sim}(1^\kappa,f_\kappa)} [\mathcal{D}_\kappa(1^\kappa,\mathsf{crs},\pi)=1] \right| < \epsilon_w(\kappa).$$

If the above holds also for unbounded circuit families $\mathcal{D}$ *then we say that* Π *is* $\epsilon_w(\kappa)$*-statistically-WI.*

Definition A.10 (Non-interactive zero-knowledge)**.** *Let* Π *be an NIP that satisfies completeness. We say that* Π *is* $(\epsilon_s(\kappa), \epsilon_z(\kappa))$*-weak NIZK if it satisfies* $\epsilon_s(\kappa)$*-soundness and* $\epsilon_z(\kappa)$*-zero knowledge. It is a standard NIZK if it satisfies* $\mathsf{negl}(\kappa)$*-soundness and* $\mathsf{negl}(\kappa)$*-zero knowledge.*

Definition A.11 (Proof of knowledge)**.** *An NIP* $\Pi = (\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ *is said to be* $\epsilon_p(\kappa)$-proof of knowledge *if there exist a PPT algorithm* Gen' *and an efficient deterministic algorithm* Ext *that satisfy the following properties:*

- (CRS indistinguishability) $\mathsf{Gen}'(1^\kappa)$ *takes a security parameter* 1^κ *and outputs* (crs', s') *such that* crs' *has the same distribution as* $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\kappa)$*.*
- (Extraction) *For every polynomial-sized non-uniform circuit family of malicious provers* $\mathcal{P}^* = \{\mathcal{P}^*_\kappa\}_{\kappa\in\mathbb{N}}$ *and all sufficiently large* $\kappa \in \mathbb{N}$*,*

$$\Pr_{\substack{(\mathsf{crs}',s')\leftarrow\mathsf{Gen}'(1^\kappa)\\ (f,\pi)\leftarrow\mathcal{P}^*_\kappa(\mathsf{crs}')}} [\mathsf{Ext}(1^\kappa,\mathsf{crs}',s',f,\pi) = \mathbf{x} \wedge f(\mathbf{x}) = 0 \wedge \mathcal{V}(1^\kappa,\mathsf{crs}',f,\pi) = 1] \leq \epsilon_p(\kappa).$$

We may omit $\epsilon_p(\kappa)$ *when it is negligible.*

A.2.2 Public-Key Encryption

The following is taken from [BG24].

Definition A.12 (Public-key encryption.)**.** *A* public-key encryption *scheme is a tuple* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *of PPT's that satisfy the following properties:*

- **Correctness:** *With all but negligible probability over the choice* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa)$*, for all* $m \in \{0,1\}^*$*, it holds that*
$$\Pr[\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(m)) = m] = 1,$$
where the probability is taken over the random coins of the encryption algorithm.
- **Privacy:** *For every ensemble of polynomial-sized messages* $\{m_\kappa\}_{\kappa\in\mathbb{N}}$ *for every family of polynomial-sized non-uniform circuits* $\mathcal{D}^* = \{\mathcal{D}^*_\kappa\}_{\kappa\in\mathbb{N}}$*, there exists a negligible function* μ *such that for all sufficiently large* κ*,*
$$\left|\Pr[\mathcal{D}^*_\kappa(\mathsf{pk}, \mathsf{Enc}_{\mathsf{sk}}(m_\kappa)) = 1] - \Pr[\mathcal{D}^*_\kappa(\mathsf{pk}, \mathsf{Enc}_{\mathsf{sk}}(0^{|m_\kappa|})) = 1]\right| \leq \mu(\kappa),$$
where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa)$*.*

In the definition, we assume that the scheme is almost-all-keys perfectly-correct, as public-key encryption schemes can be immunized [DNR04].

B Missing Proofs: Leakage Resilient MPC

In this section we provide missing proofs regarding leakage resilient MPC. We begin with the following useful claim that shows that if we use an LRSS to share many bits among the parties, the result is still leakage resilient. To simplify notation, for vectors $(\mathbf{b}_0^1, \ldots, \mathbf{b}_0^\ell)$ and $(\mathbf{b}_1^1, \ldots, \mathbf{b}_1^\ell)$, for leakage functions τ_0, τ_1 and for a set $Z \subsetneq \{0,1\}$ we first define

$$\mathsf{vLeak}_{Z,\tau_0,\tau_1}((\mathbf{b}_0^i, \mathbf{b}_1^i)_{i\in[\ell]}) := \left((\mathbf{b}_j^i)_{i\in[\ell],j\in Z}, (\tau_j((\mathbf{b}_j^i)_{i\in[\ell]}))_{j\notin Z}\right).$$

Claim B.1. *Let* $\mathsf{LRSS} = (\mathsf{LRShare}, \mathsf{LRRecover})$ *be a two-party* (μ,ϵ)*-local leakage resilient scheme for sharing a single bit. Then for every pair of vectors of* ℓ *bits* $(b^1, \ldots, b^\ell) \in \{0,1\}^\ell$ *and* $(z^1, \ldots, z^\ell) \in \{0,1\}^\ell$*, for every set* $Z \subsetneq \{0,1\}$*, and every leakage functions* τ_0 *and* τ_1 *that take* ℓ LRSS *shares and return* μ *bits,*

$$\mathsf{vLeak}_{Z,\tau_0,\tau_1}((\mathbf{b}_0^i, \mathbf{b}_1^i)_{i\in[\ell]}) \equiv_{\ell\cdot\epsilon} \mathsf{vLeak}_{Z,\tau_0,\tau_1}((\mathbf{z}_0^i, \mathbf{z}_1^i)_{i\in[\ell]})$$

where $(\mathbf{b}_0^i, \mathbf{b}_1^i) \leftarrow \mathsf{LRShare}(b^i)$*, and* $(\mathbf{z}_0^i, \mathbf{z}_1^i) \leftarrow \mathsf{LRShare}(z^i)$*, for all* $i \in [\ell]$*.*

Proof. The proof follows by a standard hybrid argument. Indeed, if the statistical distance between $\mathsf{vLeak}_{Z,\tau_0,\tau_1}((\mathbf{b}_0^i, \mathbf{b}_1^i)_{i\in[\ell]})$ and $\mathsf{vLeak}_{Z,\tau_0,\tau_1}((\mathbf{z}_0^i, \mathbf{z}_1^i)_{i\in[\ell]})$ is more than $\ell\cdot\epsilon$, then there exists an index $i^* \in \{0,\ldots,\ell-1\}$ such that $\mathsf{vLeak}_{Z,\tau_0,\tau_1}((\mathbf{b}_0^i, \mathbf{b}_1^i)_{i\in[i^*]}, (\mathbf{z}_0^i, \mathbf{z}_1^i)_{i\in\{i^*+1,\ldots,\ell\}})$ is ϵ-far from $\mathsf{vLeak}_{Z,\tau_0,\tau_1}((\mathbf{b}_0^i, \mathbf{b}_1^i)_{i\in[i^*+1]}, (\mathbf{z}_0^i, \mathbf{z}_1^i)_{i\in\{i^*+2,\ldots,\ell\}})$. Necessarily $b^{i^*+1} \neq z^{i^*+1}$ or otherwise the two distributions are the same. In fact, there are fixed values $(\tilde{\mathbf{b}}_0^i, \tilde{\mathbf{b}}_1^i)_{i\in[i^*]}$ and $(\tilde{\mathbf{z}}_0^i, \tilde{\mathbf{z}}_1^i)_{i\in\{i^*+2,\ldots,\ell\}}$ such that $\mathsf{vLeak}_{Z,\tau_0,\tau_1}((\tilde{\mathbf{b}}_0^i, \tilde{\mathbf{b}}_1^i)_{i\in[i^*]}, (\mathbf{z}_0^{i^*+1}, \mathbf{z}_1^{i^*+1}), (\tilde{\mathbf{z}}_0^i, \tilde{\mathbf{z}}_1^i)_{i\in\{i^*+2,\ldots,\ell\}})$ is ϵ-far from $\mathsf{vLeak}_{Z,\tau_0,\tau_1}((\tilde{\mathbf{b}}_0^i, \tilde{\mathbf{b}}_1^i)_{i\in[i^*]}, (\mathbf{b}_0^{i^*+1}, \mathbf{b}_1^{i^*+1}), (\tilde{\mathbf{z}}_0^i, \tilde{\mathbf{z}}_1^i)_{i\in\{i^*+2,\ldots,\ell\}})$. Let τ_j' be the leakage function that takes a single LRSS share $\mathbf{s}_j$, and executes τ_j on the vector $(\tilde{\mathbf{b}}_j^i)_{i\in[i^*]}, \mathbf{s}_j, (\tilde{\mathbf{z}}_j^i)_{i\in\{i^*+2,\ldots,\ell\}}$. Then

$$\left((\mathbf{z}_j^{i^*+1})_{j\in Z}, (\tau_j'(\mathbf{z}_j^{i^*+1}))_{j\notin Z}\right) \quad \text{is } \epsilon\text{-far from} \quad \left((\mathbf{b}_j^{i^*+1})_{j\in Z}, (\tau_j'(\mathbf{b}_j^{i^*+1}))_{j\notin Z}\right),$$

in contradiction. This completes the proof of the claim. □

B.1 Proof of Theorem 4.11

Fix any inputs $\mathbf{a} \in \{0,1\}^{\ell_A}$ for Alice and $\mathbf{b} \in \{0,1\}^{\ell_B}$, any leakage functions τ_A, τ_B where τ_A takes the view of Alice, τ_B takes the view of Bob, and both functions return μ bits, any set $Z \subsetneq \{A,B\}$ and any $\mathbf{y} \in \{0,1\}^m$.

Perfect correctness. For perfect correctness, we first observe that at the input sharing stage, for every input wire of f whose value is a bit x, Alice and Bob generate a Σ-sharing $(\mathbf{x}_A, \mathbf{x}_B)$ of x. In addition, in the randomness sharing stage, for every wire outgoing from a randomness gate, Alice and Bob sample a Σ-sharing of a random bit. Let us consider a fixed choice of the Σ-shared random bits, and denote them by $\mathbf{r}$. For a wire w let us denote by $\mathsf{val}(w)$ the value of the wire w in the computation of $f(\mathbf{a}, \mathbf{b}; \mathbf{r})$. Then a standard inductive argument shows that for every wire w, Alice and Bob hold a Σ-sharing of $\mathsf{val}(w)$. This concludes the correctness analysis.

Privacy. Let us assume without loss of generality that no input wire is an output wire. We define the simulator Sim in the following way.

- **Inputs:** The simulator takes a set $Z \subsetneq \{A, B\}$.
- **Computation:** For every wire w that is not an output wire, the simulator samples $(\mathbf{w}_A, \mathbf{w}_B) \leftarrow \mathsf{LRShare}(0)$. The simulator sets $\mathsf{pView}_A = (\mathbf{w}_A)_w$ and $\mathsf{pView}_B = (\mathbf{w}_B)_w$ where we iterate over all wires w of f that are not output wires.
- **Output:** The simulator outputs $(\mathsf{pView}_A, \mathsf{pView}_B)$.

Privacy follows from Claim B.1 by observing that $(\mathsf{pView}_A, \mathsf{out}_A)$ and $(\mathsf{pView}_B, \mathsf{out}_B)$ in the real world are simply Σ-shares of some values $(v_w)_w$ over all W wires w of f (observe that since f is a randomized circuit, those values are random variables), while in the simulation view_A and view_B simply Σ-sharing of 0's. Therefore the statistical distance is at most $W \cdot \epsilon$, as required.

Real-world output independence. Fix any outputs $(\mathsf{out}_A, \mathsf{out}_B)$ in the support of $\mathsf{vShare}(f(a, b))$. Consider the random variables $(\mathsf{pView}_A, \mathsf{pView}_B)$ that are generated in an execution of π with inputs (a, b) and the random variables $(\mathsf{pView}'_A, \mathsf{pView}'_B)$ that that are generated in an execution of π with inputs (a, b) conditioned on the output being $(\mathsf{out}_A, \mathsf{out}_B)$. Observe that $(\mathsf{pView}_A, \mathsf{pView}_B)$ correspond to Σ-shares of the values of the wires in the computation of $f(a, b; \mathbf{r})$ where $\mathbf{r}$ is uniformly distributed, and that $(\mathsf{pView}'_A, \mathsf{pView}'_B)$ correspond to Σ-shares of the values of the wires in the computation of $f(a, b; \mathbf{r})$ where $\mathbf{r}$ is sampled conditioned on the output of $f(a, b; \mathbf{r})$ being $\mathsf{vREcover}(\mathsf{out}_A, \mathsf{out}_B)$. The claim now follows from Claim B.1. □

B.2 Proof of Lemma 5.2

In this section we prove Lemma 5.2.

Perfect correctness. Let us denote $\mathsf{LRRecover}_{\Sigma_1}(\mathbf{a}_0, \mathbf{a}_1) = a$, $\mathsf{LRRecover}_{\Sigma_1}(\mathbf{b}_0, \mathbf{b}_1) = b$ and $\mathsf{LRRecover}_{\Sigma_1}(\mathbf{x}_0, \mathbf{x}_1) = x$. To see that correctness holds, observe that in the step "refreshing shares" the receivers generate Σ_1-shares $\mathbf{x}'_0, \mathbf{x}'_1$ that satisfy $\mathsf{LRRecover}_{\Sigma_1}(\mathbf{x}'_0, \mathbf{x}'_1) = x$; in the step "generating matrix of evaluations", for every $\mathbf{z}_0, \mathbf{z}_1 \in \{0, 1\}^{\ell_1}$ it holds that $\mathsf{LRvRecover}_{\Sigma_2}(\vec{\mathbf{y}}_{0,0}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{0,1}^{\mathbf{z}_0,\mathbf{z}_1}) = \mathbf{y}_0^{\mathbf{z}_0,\mathbf{z}_1}$, $\mathsf{LRvRecover}_{\Sigma_2}(\vec{\mathbf{y}}_{1,0}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{1,1}^{\mathbf{z}_0,\mathbf{z}_1}) = \mathbf{y}_1^{\mathbf{z}_0,\mathbf{z}_1}$ and $\mathsf{LRRecover}_{\Sigma_1}(\mathbf{y}_0^{\mathbf{z}_0,\mathbf{z}_1}, \mathbf{y}_1^{\mathbf{z}_0,\mathbf{z}_1}) = a \cdot z \oplus b$, where $z = \mathsf{LRRecover}_{\Sigma_1}(\mathbf{z}_0, \mathbf{z}_1)$; in the step of computing the output it holds that $\mathbf{w}_0, \mathbf{w}_1$ are Σ_2-shares of the bit defined by $(\mathbf{y}_0^{\mathbf{x}'_0,\mathbf{x}'_1}, \mathbf{y}_1^{\mathbf{x}'_0,\mathbf{x}'_1})$, i.e., of $a \cdot x \oplus b$; and in the step of translating the shares, $\mathbf{m}_0, \mathbf{m}_1$ are Σ_1-shares of $a \cdot x \oplus b$.

Real-world output independence. The perfect correctness implies that $(\mathbf{m}_0, \mathbf{m}_1)$ indeed form a Σ_1-sharing of $a \cdot x \oplus b$. It is not hard to verify that they are independent of all previous execution, and have the same distribution as $\mathsf{LRShare}_{\Sigma_1}(a \cdot x \oplus b)$.

We continue with the proof of statistical privacy, where first we describe the simulator.

B.2.1 The Simulator

The simulator Sim takes as an input Z, a bit m (corresponding to out′), and aux. Let us denote by $Z' := \{S_i : \phi(S_i) \in Z\} \cup \{R_i : \phi(R_i) \in Z\}$ the set of "corrupted roles" among $\{S_0, S_1, R_0, R_1\}$. The simulator first generates views $\mathsf{pView}_{S_0}, \mathsf{pView}_{S_1}, \mathsf{pView}_{R_0}$ and pView_{R_1}, and then sets the view of every P_i to be $(\mathsf{pView}_p)_{p\in\phi^{-1}(P_i)}$. To do so, the simulator first initializes empty partial views $\mathsf{pView}_{S_0}, \mathsf{pView}_{S_1}, \mathsf{pView}_{R_0}$ and pView_{R_1}, and updates the views in the following way.

1. *(Refreshing shares)* If $R_0, R_1 \in Z'$ (that is, $\phi(R_0), \phi(R_1) \in Z$) then the simulator holds $\mathbf{x}_0, \mathbf{x}_1$ as part of aux, and the simulator updates pView_{R_0} and pView_{R_1} by executing $\mathsf{LR2PC}_{\Sigma_1}(f_{\mathsf{refresh}})$ with input $\mathbf{x}_0$ to R_0 and input $\mathbf{x}_1$ to R_1.

   Otherwise, let $\mathsf{Sim}_{\mathsf{refresh}}$ be the simulator corresponding to $\mathsf{LR2PC}_{\Sigma_1}(f_{\mathsf{refresh}})$. The simulator executes $(\mathbf{x}'_0, \mathbf{x}'_1) \leftarrow \mathsf{LRShare}_{\Sigma_1}(0)$ and samples $(\mathsf{pView}_0^{\mathsf{refresh}}, \mathsf{pView}_1^{\mathsf{refresh}}) \leftarrow \mathsf{Sim}_{\mathsf{refresh}}(Z' \cap \{R_0, R_1\})$. The simulator updates pView_{R_0} and pView_{R_1} with $(\mathsf{pView}_0^{\mathsf{refresh}}, \mathbf{x}'_0)$ and $(\mathsf{pView}_1^{\mathsf{refresh}}, \mathbf{x}'_1)$, respectively.

2. *(Generating matrix of evaluations)* If $S_0, S_1 \in Z'$ (that is, $\phi(S_0), \phi(S_1) \in Z$) then the simulator holds $(\mathbf{a}_0, \mathbf{b}_0)$ and $(\mathbf{a}_1, \mathbf{b}_1)$ as part of aux, and the simulator updates pView_{S_0} and pView_{S_1} by executing $\mathsf{LR2PC}_{\Sigma_2}(f_{\mathsf{matrix}})$ with inputs $(\mathbf{a}_0, \mathbf{b}_0)$ to S_0 and $(\mathbf{a}_1, \mathbf{b}_1)$ to S_1.

   Otherwise, let $\mathsf{Sim}_{\mathsf{matrix}}$ be the simulator corresponding to $\mathsf{LR2PC}_{\Sigma_2}(f_{\mathsf{matrix}})$. For every $\mathbf{z}_0, \mathbf{z}_1 \in \{0,1\}^{\ell_1}$ the simulator samples $(\mathbf{y}_0^{\mathbf{z}_0,\mathbf{z}_1}, \mathbf{y}_1^{\mathbf{z}_0,\mathbf{z}_1}) \leftarrow \mathsf{LRShare}_{\Sigma_1}(m)$. The simulator executes

$$((\vec{\mathbf{y}}_{0,0}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{1,0}^{\mathbf{z}_0,\mathbf{z}_1}), (\vec{\mathbf{y}}_{0,1}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{1,1}^{\mathbf{z}_0,\mathbf{z}_1}))_{\mathbf{z}_0,\mathbf{z}_1\in\{0,1\}^{\ell_1}} \leftarrow \mathsf{LRvShare}_{\Sigma_2}((\mathbf{y}_0^{\mathbf{z}_0,\mathbf{z}_1}, \mathbf{y}_1^{\mathbf{z}_0,\mathbf{z}_1})_{\mathbf{z}_0,\mathbf{z}_1\in\{0,1\}^{\ell_1}}),$$

   and samples $(\mathsf{pView}_0^{\mathsf{matrix}}, \mathsf{pView}_1^{\mathsf{matrix}}) \leftarrow \mathsf{Sim}_{\mathsf{matrix}}(Z' \cap \{S_0, S_1\})$. The simulator updates pView_{S_0} with $(\mathsf{pView}_0^{\mathsf{matrix}}, (\vec{\mathbf{y}}_{0,0}^{\mathbf{z}_1,\mathbf{z}_0}, \vec{\mathbf{y}}_{1,0}^{\mathbf{z}_0,\mathbf{z}_1})_{\mathbf{z}_0,\mathbf{z}_1\in\{0,1\}^{\ell_1}})$ and pView_{S_1} with $(\mathsf{pView}_1^{\mathsf{matrix}}, (\vec{\mathbf{y}}_{0,1}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{1,1}^{\mathbf{z}_0,\mathbf{z}_1})_{\mathbf{z}_0,\mathbf{z}_1\in\{0,1\}^{\ell_1}})$, respectively.

3. *(Transmitting $\mathbf{x}'_0$-row to R_0)* The outputs of the $\mathcal{F}_{\mathbf{x}'_0}^{\mathsf{condTrans}}$ calls are set to be $(\vec{\mathbf{y}}_{0,0}^{\mathbf{x}'_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{0,1}^{\mathbf{x}'_0,\mathbf{z}_1})_{\mathbf{z}_1\in\{0,1\}^{\ell_1}}$ and the rest of the calls, for $\mathcal{F}_{\mathbf{z}_0}^{\mathsf{condTrans}}$ with $\mathbf{z}_0 \neq \mathbf{x}'_0$, the output is set to be 0. We then recover $(\mathbf{y}_0^{\mathbf{x}'_0,\mathbf{z}_1})_{\mathbf{z}_1\in\{0,1\}^{\ell_1}}$.

4. *(Transmitting $\mathbf{x}'_1$-column to R_1)* The outputs of the $\mathcal{F}_{\mathbf{x}'_1}^{\mathsf{condTrans}}$ calls are set to be $(\vec{\mathbf{y}}_{1,0}^{\mathbf{z}_0,\mathbf{x}'_1}, \vec{\mathbf{y}}_{1,1}^{\mathbf{z}_0,\mathbf{x}'_1})_{\mathbf{z}_0\in\{0,1\}^{\ell_1}}$ and the rest of the calls, for $\mathcal{F}_{\mathbf{z}_1}^{\mathsf{condTrans}}$ with $\mathbf{z}_1 \neq \mathbf{x}'_1$, the output is set to be 0. We then recover $(\mathbf{y}_1^{\mathbf{z}_0,\mathbf{x}'_1})_{\mathbf{z}_0\in\{0,1\}^{\ell_1}}$.

5. *(Computing the output)* If $R_0, R_1 \in Z'$ (i.e., $\phi(R_0), \phi(R_1) \in Z$) then the simulator, that holds $\mathbf{x}'_0, \mathbf{x}'_1$, updates pView_{R_0} and pView_{R_1} by executing $\mathsf{LR2PC}_{\Sigma_2}(f_{\mathsf{out}})$ with input $\mathbf{x}'_0, (\mathbf{y}_0^{\mathbf{x}'_0,\mathbf{z}_1})_{\mathbf{z}_1\in\{0,1\}^{\ell_1}}$ to R_0 and input $\mathbf{x}'_1, (\mathbf{y}_0^{\mathbf{z}_0,\mathbf{x}'_1})_{\mathbf{z}_0\in\{0,1\}^{\ell_1}}$ to R_1.

   Otherwise, let $\mathsf{Sim}_{f_{\mathsf{out}}}$ be the simulator corresponding to $\mathsf{LR2PC}_{\Sigma_2}(f_{\mathsf{out}})$. The simulator executes $(\mathbf{w}_0, \mathbf{w}_1) \leftarrow \mathsf{LRShare}_{\Sigma_2}(m)$, samples $(\mathsf{pView}_0^{\mathsf{out}}, \mathsf{pView}_1^{\mathsf{out}}) \leftarrow \mathsf{Sim}_{f_{\mathsf{out}}}(Z' \cap \{R_0, R_1\})$. The simulator updates pView_{R_0} and pView_{R_1} with $(\mathsf{pView}_0^{\mathsf{out}}, \mathbf{w}_0)$ and $(\mathsf{pView}_1^{\mathsf{out}}, \mathbf{w}_1)$, respectively.

6. *(Translating the shares)* In this step the parties compute their outputs $\mathbf{m}_0$ and $\mathbf{m}_1$. Since the outputs are not part of the partial view, this step is not required to be simulated.

The simulator outputs $(\mathsf{pView}_{P_1}, \ldots, \mathsf{pView}_{P_n})$ where $\mathsf{pView}_{P_i} := (\mathsf{pView}_p)_{p\in\phi^{-1}(P_i)}$.

B.2.2 Analysis

Fix inputs $(\mathbf{a}_0, \mathbf{b}_0)$, $(\mathbf{a}_1, \mathbf{b}_1)$, $\mathbf{x}_0$ and $\mathbf{x}_1$, leakage functions $\tau_1, \ldots, \tau_n$ that output μ' bits, a bit $\beta \in \{0,1\}$ and a set $Z \subseteq \{P_1, \ldots, P_n\}$, and let $Z' := \{S_i : \phi(S_i) \in Z\} \cup \{R_i : \phi(R_i) \in Z\}$. Let us denote $a = \mathsf{LRRecover}_{\Sigma_1}(\mathbf{a}_0, \mathbf{a}_1)$, $b = \mathsf{LRRecover}_{\Sigma_1}(\mathbf{b}_0, \mathbf{b}_1)$ and $x = \mathsf{LRRecover}_{\Sigma_1}(\mathbf{x}_0, \mathbf{x}_1)$. Observe that $m := a \cdot x \oplus b$ if $R_0, R_1 \in Z'$, and $m := \beta$ otherwise (where m corresponds to the bit out' that the simulator takes as an input). We also denote by $\mathsf{size}(\mathsf{refresh})$, $\mathsf{size}(\mathsf{matrix})$ and $\mathsf{size}(\mathsf{out})$ the size of the circuits f_{refresh}, f_{matrix} and f_{out}, respectively, measured by the number of wires.

In the following we define hybrids, where each hybrid generates partial views $\mathsf{pView}_{S_0}, \mathsf{pView}_{S_1}, \mathsf{pView}_{R_0}$ and pView_{R_1}, together with outputs of the receivers, out_{R_0} and out_{R_1}. The output of the hybrid is set to be $(\mathsf{pView}_{P_i}, \mathsf{out}_{P_i})_{P_i \in Z}, (\tau_{P_i}(\mathsf{pView}_{P_i}, \mathsf{out}_{P_i}))_{P_i \in \mathcal{P} \setminus Z}$, where $\mathsf{pView}_{P_i} := (\mathsf{pView}_p)_{p \in \phi^{-1}(P_i)}$ and $\mathsf{out}_{P_i} := \mathsf{out}_{R_j}$ if $\phi(R_j) = P_i$, and otherwise, if $\phi^{-1}(P_i)$ contains only a sender, then $\mathsf{out}_{P_i} = \perp$. We continue by explaining how the hybrids generate $\mathsf{pView}_{S_0}, \mathsf{pView}_{S_1}, \mathsf{pView}_{R_0}, \mathsf{pView}_{R_1}, \mathsf{out}_{R_0}$ and out_{R_1}.

Hybrid 1. This Hybrid is equivalent to $\mathsf{pIdeal}_{\mathsf{LROLE},Z,\vec{\tau}}(((\mathbf{a}_0, \mathbf{b}_0), (\mathbf{a}_1, \mathbf{b}_1)), (\mathbf{x}_0, \mathbf{x}_1), \beta)$. That is, if $R_0, R_1 \in Z'$ then we set $m := a \cdot x \oplus b$ and otherwise $m := \beta$. Then, we sample the outputs of the receivers by $(\mathsf{out}_{R_0}, \mathsf{out}_{R_1}) \leftarrow \mathsf{LRShare}_{\Sigma_1}(m)$, and the partial views $\mathsf{pView}_{S_0}, \mathsf{pView}_{S_1}, \mathsf{pView}_{R_0}, \mathsf{pView}_{R_1}$ are generated just like in $\mathsf{Sim}(Z, m, \mathsf{aux})$, where aux contains $(\mathbf{a}_0, \mathbf{a}_1, \mathbf{b}_0, \mathbf{b}_1)$ if $S_0, S_1 \in Z'$, and $(\mathbf{x}_0, \mathbf{x}_1)$ if $R_0, R_1 \in Z'$.

Hybrid 2. In Hybrid 2, the outputs and partial views are generated like in Hybrid 1 with the following modifications. The step of "refreshing shares" (Step (1)) is simulated by letting R_0 and R_1 execute $\mathsf{LR2PC}_{\Sigma_1}(f_{\mathsf{refresh}})$ on inputs $(\mathbf{x}_0, \mathbf{x}_1)$.

Hybrid 3. In Hybrid 3, the outputs and partial views are generated like in Hybrid 2 with the following modifications. The step of "generating matrix of evaluations" (Step (2)) is simulated by letting S_0 and S_1 execute $\mathsf{LR2PC}_{\Sigma_2}(f_{\mathsf{matrix}})$ on inputs $(\mathbf{a}_0, \mathbf{b}_0)$ and $(\mathbf{a}_1, \mathbf{b}_1)$.

Hybrid 4. In Hybrid 4, the outputs and partial views are generated like in Hybrid 3 with the following modifications. The steps of "transmitting $\mathbf{x}'_0$-row to R_0" (Step (3)) and "transmitting $\mathbf{x}'_1$-column to R_1" (Step (4)) is executed like in the protocol.

Hybrid 5. In Hybrid 5, the outputs and partial views are generated like in Hybrid 4 with the following modifications. The steps of "computing the output" (Step (5)) is executed like in the protocol.

Hybrid 6. Hybrid 6 is $\mathsf{pReal}_{\mathsf{LROLE},Z,\vec{\tau}}((\mathbf{a}_0, \mathbf{b}_0), (\mathbf{a}_1, \mathbf{b}_1), \mathbf{x}_0, \mathbf{x}_1)$.

To prove Lemma 5.2, it is enough to prove the following claims. The claims are proved in Sections (B.2.3)–(B.2.6), except for Claim B.4 whose proof is straightforward.

Claim B.2. *It holds that* $\mathsf{hybrid}_1 \equiv_{\mathsf{size}(\mathsf{refresh}) \cdot \epsilon_1} \mathsf{hybrid}_2$.

Claim B.3. *It holds that* $\mathsf{hybrid}_2 \equiv_{\epsilon_1 + 2 \cdot 2^{2\ell_1} \cdot \ell_1 \cdot \epsilon_2 + 2 \cdot \mathsf{size}(\mathsf{matrix}) \cdot \epsilon_2} \mathsf{hybrid}_3$.

Claim B.4. *It holds that* $\mathsf{hybrid}_3 \equiv \mathsf{hybrid}_4$.

Claim B.5. *It holds that* $\mathsf{hybrid}_4 \equiv_{\mathsf{size(out)}\cdot\epsilon_2} \mathsf{hybrid}_5$.

Claim B.6. *It holds that* $\mathsf{hybrid}_5 \equiv_{\epsilon_1} \mathsf{hybrid}_6$.

B.2.3 Proof of Claim B.2

Let $Z_R := Z' \cap \{R_0, R_1\}$. It is not hard to see that if $Z_R = \{R_0, R_1\}$ then $\mathsf{hybrid}_1 \equiv \mathsf{hybrid}_2$. Therefore, let us assume that $Z_R \subsetneq \{R_0, R_1\}$. Assume towards contradiction that hybrid_1 is $\mathsf{size(refresh)} \cdot \epsilon_1$-far from hybrid_2. We show that there exist two leakage functions ν_{R_0}, ν_{R_1} that output μ' bits, such that

$$\mathsf{pReal}_{\mathsf{LR2PC}_{\Sigma_1}(f_{\mathsf{refresh}}),Z_R,\vec{\nu}}(\mathbf{x}_0,\mathbf{x}_1) \quad \text{is } \mathsf{size(refresh)} \cdot \epsilon_1\text{-far from} \quad \mathsf{pIdeal}_{\mathsf{LR2PC}_{\Sigma_1}(f_{\mathsf{refresh}}),Z_R,\vec{\nu}}(0).$$

in contradiction to the strong $(\mu, \mathsf{size(refresh)} \cdot \epsilon_1)$-composable leakage-resilient passive security of $\mathsf{LR2PC}_{\Sigma_1}(f_{\mathsf{refresh}})$ (see Theorem 4.11). To see this, we first observe that the following steps in the simulator are independent of $(\mathsf{pView}_0^{\mathsf{refresh}}, \mathbf{x}'_0)$ and $(\mathsf{pView}_1^{\mathsf{refresh}}, \mathbf{x}'_1)$: Step (2) of generating matrix of evaluations, Step (5) of computing the output, and the simulation of the outputs $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathsf{LRShare}_{\Sigma_1}(m)$. Therefore, in both hybrids these steps have the same distribution, so there exist fixed values of $(\mathsf{pView}_0^{\mathsf{matrix}}, (\vec{\mathbf{y}}_{0,0}^{\mathbf{z}_1,\mathbf{z}_0}, \vec{\mathbf{y}}_{1,0}^{\mathbf{z}_0,\mathbf{z}_1})_{\mathbf{z}_0,\mathbf{z}_1\in\{0,1\}^{\ell_1}})$ and $(\mathsf{pView}_1^{\mathsf{matrix}}, (\vec{\mathbf{y}}_{0,1}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{1,1}^{\mathbf{z}_0,\mathbf{z}_1})_{\mathbf{z}_0,\mathbf{z}_1\in\{0,1\}^{\ell_1}})$ from Step (2), $(\mathsf{pView}_0^{\mathsf{out}}, \mathbf{w}_0)$ and $(\mathsf{pView}_1^{\mathsf{out}}, \mathbf{w}_1)$ from Step (5), and the simulated outputs $(\mathbf{m}_0, \mathbf{m}_1)$, such that hybrid_1 is $\mathsf{size(refresh)}\cdot\epsilon_1$-far from hybrid_2 even conditioned on those values. Let us denote the conditional distributions by hybrid'_1 and hybrid'_2. Observe that the partial views corresponding to S_0 and S_1 are fixed, and denote the fixed views by pView'_{S_0} and pView'_{S_1}.

View completion procedure and leakage functions. We define the view completion procedure for R_i to be the procedure that takes as an input $(\mathsf{pView}_i^{\mathsf{refresh}}, \mathbf{x}'_i)$ and computes the partial view of R_i in LROLE, denoted pView'_{R_i}, by concatenating the fixed values $(\vec{\mathbf{y}}_{0,0}^{\mathbf{x}'_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{0,1}^{\mathbf{x}'_0,\mathbf{z}_1})_{\mathbf{z}_1\in\{0,1\}^{\ell_1}}$ if $i = 0$, or $(\vec{\mathbf{y}}_{1,0}^{\mathbf{z}_0,\mathbf{x}'_1}, \vec{\mathbf{y}}_{1,1}^{\mathbf{z}_0,\mathbf{x}'_1})_{\mathbf{z}_0\in\{0,1\}^{\ell_1}}$ if $i = 1$ (and zero from the remaining $\mathcal{F}^{\mathsf{condTrans}}$ calls), as well as the fixed values $(\mathsf{pView}_i^{\mathsf{out}}, \mathbf{w}_i)$.

The new leakage functions ν_{R_0} and ν_{R_1} have the partial views $(\mathsf{pView}'_{S_0}, \mathsf{pView}'_{S_1})$ and the outputs $(\mathbf{m}_0, \mathbf{m}_1)$ hardwired to them, and are defined in the following way: ν_{R_i} takes as an input $(\mathsf{pView}_i^{\mathsf{refresh}}, \mathbf{x}'_i)$ executes the view completion procedure to obtain pView'_{R_i}, and outputs $\tau_{\phi(R_i)}((\mathsf{pView}'_p)_{p\in\phi^{-1}(\phi(R_i))}, \mathbf{m}_i)$. (Here we use the fact that $\phi(R_0) \neq \phi(R_1)$.)

Analysis. Let us define pReal' in the following way:

1. Sample
$$(\mathsf{pView}_i^{\mathsf{refresh}}, \mathbf{x}'_i)_{R_i\in Z_R}, (\nu_{R_i}(\mathsf{pView}_i^{\mathsf{refresh}}, \mathbf{x}'_i))_{R_i\notin Z_R}$$
as in $\mathsf{pReal}_{\mathsf{LR2PC}_\Sigma(f_{\mathsf{refresh}}),Z_R,\vec{\nu}}(\mathbf{x}_0,\mathbf{x}_1)$.
2. For every $R_i \in Z_R$, execute the view completion procedure to obtain pView_{R_i}.
3. For every $R_i \in Z_R$, define $\mathsf{pView}_{\phi(R_i)} := (\mathsf{pView}'_p)_{p\in\phi^{-1}(\phi(R_i))}$ and $\mathsf{out}_{\phi(R_i)} := \mathbf{m}_i$.
4. For every $R_i \notin Z_R$, define $\mathsf{leak}_{\phi(R_i)} := \nu_{R_i}(\mathsf{pView}_i^{\mathsf{refresh}}, \mathbf{x}'_i)$.

5. For every $P_i \in \mathcal{P} \setminus \{\phi(R_0), \phi(R_1)\}$ define $\mathsf{pView}_{P_i} := (\mathsf{pView}'_{S_j})_{S_j \in \phi^{-1}(P_i)}$ and $\mathsf{out}_{P_i} := \perp$. If $P_i \notin Z$ then define $\mathsf{leak}_{P_i} := \tau_{P_i}(\mathsf{pView}_{P_i})$.

6. Output $(\mathsf{pView}_{P_i}, \mathsf{out}_{P_i})_{P_i \in Z}, (\mathsf{leak}_{P_i})_{P_i \notin Z}$.

Define pIdeal' analogously, where the only difference is that step (1) is executed with respect to $\mathsf{pIdeal}_{\mathsf{LR2PC}(f_{\mathsf{refresh}}),Z_R,\vec{\nu}}(0)$ instead of $\mathsf{pReal}_{\mathsf{LR2PC}_\Sigma(f_{\mathsf{refresh}}),Z_R,\vec{\nu}}(\mathbf{x}_0, \mathbf{x}_1)$.

It is not hard to verify that $\mathsf{pReal}' \equiv \mathsf{hybrid}'_2$ and $\mathsf{pIdeal}' \equiv \mathsf{hybrid}'_1$. Therefore, pReal' is $\mathsf{size}(\mathsf{refresh}) \cdot \epsilon_1$-far from pIdeal'. In addition, since pReal' and pIdeal' are obtained from $\mathsf{pReal}_{\mathsf{LR2PC}_\Sigma(f_{\mathsf{refresh}}),Z_R,\vec{\nu}}(\mathbf{x}_0, \mathbf{x}_1)$ and $\mathsf{pIdeal}_{\mathsf{LR2PC}(f_{\mathsf{refresh}}),Z_R,\vec{\nu}}(0)$ by the same procedure, we obtain that $\mathsf{pReal}_{\mathsf{LR2PC}_\Sigma(f_{\mathsf{refresh}}),Z_R,\vec{\nu}}(\mathbf{x}_0, \mathbf{x}_1)$ and $\mathsf{pIdeal}_{\mathsf{LR2PC}(f_{\mathsf{refresh}}),Z_R,\vec{\nu}}(0)$ are $\mathsf{size}(\mathsf{refresh}) \cdot \epsilon_1$-far, in contradiction. This concludes the proof of the claim. □

B.2.4 Proof of Claim B.3

Let us denote $Z_S := Z' \cap \{S_0, S_1\}$ and $Z_R := Z' \cap \{R_0, R_1\}$. It is not hard to see that if $Z_S = \{S_0, S_1\}$ then $\mathsf{hybrid}_2 \equiv \mathsf{hybrid}_3$. Therefore, let us assume that $Z_S \subsetneq \{S_0, S_1\}$. To prove the claim, we first define the following intermediate hybrid.

Hybrid 2.1. Hybrid 2.1 is identical to Hybrid 2, except that in the step "generating matrix of evaluations" of the simulator (Step (2)), we sample $(\mathbf{y}_0^{\mathbf{x}'_0,\mathbf{x}'_1}, \mathbf{y}_1^{\mathbf{x}'_0,\mathbf{x}'_1}) \leftarrow \mathsf{LRShare}_{\Sigma_1}(a \cdot x \oplus b)$, while the remaining pairs $(\mathbf{y}_0^{\mathbf{z}_0,\mathbf{z}_1}, \mathbf{y}_1^{\mathbf{z}_0,\mathbf{z}_1})$ are sampled like in Hybrid 2 from $\mathsf{LRShare}_{\Sigma_1}(m)$. Observe that those random variables can be sampled after the step "refreshing shares" in which $(\mathbf{x}'_0, \mathbf{x}'_1)$ is defined. After that, the outputs $((\vec{\mathbf{y}}_{0,0}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{1,0}^{\mathbf{z}_0,\mathbf{z}_1}), (\vec{\mathbf{y}}_{0,1}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{1,1}^{\mathbf{z}_0,\mathbf{z}_1}))_{\mathbf{z}_0,\mathbf{z}_1 \in \{0,1\}^{\ell_1}}$ are sampled from $\mathsf{LRvShare}_{\Sigma_2}((\mathbf{y}_0^{\mathbf{z}_0,\mathbf{z}_1}, \mathbf{y}_1^{\mathbf{z}_0,\mathbf{z}_1})_{\mathbf{z}_0,\mathbf{z}_1 \in \{0,1\}^{\ell_1}})$ like in Hybrid 2.

Hybrid 2.2. Hybrid 2.2 is identical to Hybrid 2.1, except that for every $(\mathbf{z}_0, \mathbf{z}_1)$, we sample $(\mathbf{y}_0^{\mathbf{z}_0,\mathbf{z}_1}, \mathbf{y}_1^{\mathbf{z}_0,\mathbf{z}_1})$ by first computing the bit $z = \mathsf{LRRecover}(\mathbf{z}_0, \mathbf{z}_1)$ and then sampling $(\mathbf{y}_0^{\mathbf{z}_0,\mathbf{z}_1}, \mathbf{y}_1^{\mathbf{z}_0,\mathbf{z}_1}) \leftarrow \mathsf{LRShare}_{\Sigma_1}(a \cdot z \oplus b)$. After that, the outputs $((\vec{\mathbf{y}}_{0,0}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{1,0}^{\mathbf{z}_0,\mathbf{z}_1}), (\vec{\mathbf{y}}_{0,1}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{1,1}^{\mathbf{z}_0,\mathbf{z}_1}))_{\mathbf{z}_0,\mathbf{z}_1 \in \{0,1\}^{\ell_1}}$ are sampled from $\mathsf{LRvShare}_{\Sigma_2}((\mathbf{y}_0^{\mathbf{z}_0,\mathbf{z}_1}, \mathbf{y}_1^{\mathbf{z}_0,\mathbf{z}_1})_{\mathbf{z}_0,\mathbf{z}_1 \in \{0,1\}^{\ell_1}})$.

Hybrid 2.3. Hybrid 2.3 is identical to Hybrid 2.2, except that $(\mathsf{pView}_0^{\mathsf{matrix}}, \mathsf{pView}_1^{\mathsf{matrix}})$ are computed by executing $\mathsf{LR2PC}_{\Sigma_2}(f_{\mathsf{matrix}})$ where S_0 holds $(\mathbf{a}_0, \mathbf{b}_0)$ and S_1 holds $(\mathbf{a}_1, \mathbf{b}_1)$. After that, the outputs $((\vec{\mathbf{y}}_{0,0}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{1,0}^{\mathbf{z}_0,\mathbf{z}_1}), (\vec{\mathbf{y}}_{0,1}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{1,1}^{\mathbf{z}_0,\mathbf{z}_1}))_{\mathbf{z}_0,\mathbf{z}_1 \in \{0,1\}^{\ell_1}}$ are sampled like in Hybrid 2.2, independently of the execution of $\mathsf{LR2PC}_{\Sigma_2}(f_{\mathsf{matrix}})$.

To prove Claim B.3 it is enough to prove the following claims.

Claim B.7. $\mathsf{hybrid}_2 \equiv_{\epsilon_1} \mathsf{hybrid}_{2.1}$.

Claim B.8. $\mathsf{hybrid}_{2.1} \equiv_{2 \cdot 2^{2\ell_1} \cdot \ell_1 \cdot \epsilon_2} \mathsf{hybrid}_{2.2}$.

Claim B.9. $\mathsf{hybrid}_{2.2} \equiv_{\mathsf{size}(\mathsf{matrix}) \cdot \epsilon_2} \mathsf{hybrid}_{2.3}$.

Claim B.10. $\mathsf{hybrid}_{2.3} \equiv_{\mathsf{size}(\mathsf{matrix}) \cdot \epsilon_2} \mathsf{hybrid}_3$.

We continue with the proofs of the claims.

Proof of Claim B.7. It is not hard to see that $\mathsf{hybrid}_2 \equiv \mathsf{hybrid}_{2.1}$ if $Z_R = \{R_0, R_1\}$, since in this case $m = a \cdot x \oplus b$. Therefore, let us assume that $Z_R \subsetneq \{R_0, R_1\}$.

Focusing on the two-party case. First, we focus on the special case where there are only two-parties, P_1 and P_2. By symmetry, we may assume without loss of generality that $\phi(S_0) = \phi(R_0) = P_1$ and $\phi(S_1) = \phi(R_1) = P_2$. We may also assume that $Z = \{P_1\}$, so $Z' = \{S_0, R_0\}$ (indeed, adding parties for the set Z may only increase the statistical distance between the hybrids). In fact, for this special case we prove that $\mathsf{hybrid}_2 \equiv_{\epsilon_1} \mathsf{hybrid}_{2.1}$ even when the underlying leakage functions τ_1, τ_2 are allowed to leak $2\mu'$ bits instead of just μ' bits. At the end of the proof we explain how to reduce the general case to this special case.

Let us denote the hybrids where the leakage functions are allowed to leak $2\mu'$ bits by hybrid'_2 and $\mathsf{hybrid}'_{2.1}$. Assume towards contradiction that hybrid'_2 is ϵ_1-far from $\mathsf{hybrid}'_{2.1}$.

Fixing non-relevant variables. We first observe that the step of "refreshing shares" in both hybrids is executed in the same way, and therefore the random variables $(\mathsf{pView}_0^{\mathsf{refresh}}, \mathbf{x}'_0)$ and $(\mathsf{pView}_1^{\mathsf{refresh}}, \mathbf{x}'_1)$ have the same distribution in both hybrids. Therefore, there exists a fixed choice of values to those random variables, such that hybrid'_2 is ϵ_1-far from $\mathsf{hybrid}'_{2.1}$ even conditioned on those values. In addition, in the step of "generating matrix of evaluations" the random variables $(\mathsf{pView}_0^{\mathsf{matrix}}, \mathsf{pView}_1^{\mathsf{matrix}})$ and $(\vec{\mathbf{y}}_{0,0}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{0,1}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{1,0}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{1,1}^{\mathbf{z}_0,\mathbf{z}_1})_{(\mathbf{z}_0,\mathbf{z}_1)\neq(\mathbf{x}'_0,\mathbf{x}'_1)}$ are sampled in the same way in both distributions, and this is also true for the steps "computing the output" and the outputs with respect to the random variables $(\mathsf{pView}_0^{\mathsf{out}}, \mathbf{w}_0)$, $(\mathsf{pView}_1^{\mathsf{out}}, \mathbf{w}_1)$, $\mathbf{m}_0$ and $\mathbf{m}_1$. Therefore there is a fixed choice of them such that hybrid'_2 is ϵ_1-far from $\mathsf{hybrid}'_{2.1}$ even conditioned on those values. Denote the hybrids conditioned on those fixed values by hybrid''_2 and $\mathsf{hybrid}''_{2.1}$.

Completing views. In both hybrids, the only random variables that are not fixed are $(\vec{\mathbf{y}}_{0,0}^{\mathbf{x}'_0,\mathbf{x}'_1}, \vec{\mathbf{y}}_{0,1}^{\mathbf{x}'_0,\mathbf{x}'_1}, \vec{\mathbf{y}}_{1,0}^{\mathbf{x}'_0,\mathbf{x}'_1}, \vec{\mathbf{y}}_{1,1}^{\mathbf{x}'_0,\mathbf{x}'_1})$ (together with the induced values $\mathbf{y}_0^{\mathbf{x}'_0,\mathbf{x}'_1}, \mathbf{y}_1^{\mathbf{x}'_0,\mathbf{x}'_1}$). In hybrid''_2 it holds that

$$(\mathbf{y}_0^{\mathbf{x}'_0,\mathbf{x}'_1}, \mathbf{y}_1^{\mathbf{x}'_0,\mathbf{x}'_1}) \leftarrow \mathsf{LRShare}_{\Sigma_1}(m) \quad \text{and} \quad ((\vec{\mathbf{y}}_{0,0}^{\mathbf{x}'_0,\mathbf{x}'_1}, \vec{\mathbf{y}}_{1,0}^{\mathbf{x}'_0,\mathbf{x}'_1}), (\vec{\mathbf{y}}_{0,1}^{\mathbf{x}'_0,\mathbf{x}'_1}, \vec{\mathbf{y}}_{1,1}^{\mathbf{x}'_0,\mathbf{x}'_1})) \leftarrow \mathsf{LRvShare}_{\Sigma_2}(\mathbf{y}_0^{\mathbf{x}'_0,\mathbf{x}'_1}, \mathbf{y}_1^{\mathbf{x}'_0,\mathbf{x}'_1}),$$

and in $\mathsf{hybrid}''_{2.1}$ it holds that

$$(\mathbf{y}_0^{\mathbf{x}'_0,\mathbf{x}'_1}, \mathbf{y}_1^{\mathbf{x}'_0,\mathbf{x}'_1}) \leftarrow \mathsf{LRShare}_{\Sigma_1}(a{\cdot}x{\oplus}b) \quad \text{and} \quad ((\vec{\mathbf{y}}_{0,0}^{\mathbf{x}'_0,\mathbf{x}'_1}, \vec{\mathbf{y}}_{1,0}^{\mathbf{x}'_0,\mathbf{x}'_1}), (\vec{\mathbf{y}}_{0,1}^{\mathbf{x}'_0,\mathbf{x}'_1}, \vec{\mathbf{y}}_{1,1}^{\mathbf{x}'_0,\mathbf{x}'_1})) \leftarrow \mathsf{LRvShare}_{\Sigma_2}(\mathbf{y}_0^{\mathbf{x}'_0,\mathbf{x}'_1}, \mathbf{y}_1^{\mathbf{x}'_0,\mathbf{x}'_1}).$$

Since all remaining values are fixed, given $(\vec{\mathbf{y}}_{0,0}^{\mathbf{x}'_0,\mathbf{x}'_1}, \vec{\mathbf{y}}_{1,0}^{\mathbf{x}'_0,\mathbf{x}'_1})$ we can naturally compute S_0's view, given $(\vec{\mathbf{y}}_{0,1}^{\mathbf{x}'_0,\mathbf{x}'_1}, \vec{\mathbf{y}}_{1,1}^{\mathbf{x}'_0,\mathbf{x}'_1})$ we can compute S_1's view, given $(\vec{\mathbf{y}}_{0,0}^{\mathbf{x}'_0,\mathbf{x}'_1}, \vec{\mathbf{y}}_{0,1}^{\mathbf{x}'_0,\mathbf{x}'_1})$ we can compute R_0's view and output, and given $(\vec{\mathbf{y}}_{1,0}^{\mathbf{x}'_0,\mathbf{x}'_1}, \vec{\mathbf{y}}_{1,1}^{\mathbf{x}'_0,\mathbf{x}'_1})$ we can compute R_1's view and output.

Breaking Σ_1. Since hybrid''_2 is ϵ_1-far from $\mathsf{hybrid}''_{2.1}$ then necessarily $a \cdot x \oplus b \neq m$. Let us assume without loss of generality that $a \cdot x \oplus b = 0$ and therefore $m = 1$. We will show that we can distinguish a Σ_1-sharing of 0 from a Σ_1-sharing of 1.

Let $\mathbf{r} = (\vec{\mathbf{r}}_{1,0}, \vec{\mathbf{r}}_{0,1}, \vec{\mathbf{r}}'_{0,0}, \vec{\mathbf{r}}'_{1,1})$ be a random tuple that is sampled in the following way: the vectors $\vec{\mathbf{r}}_{1,0}$ and $\vec{\mathbf{r}}_{0,1}$ are of length ℓ_1, and their i-th element is a random Σ_2-share, i.e., $\vec{\mathbf{r}}_{1,0}[i] \leftarrow \mathsf{LRRand}_{\Sigma_2}$ and $\vec{\mathbf{r}}_{0,1}[i] \leftarrow \mathsf{LRRand}_{\Sigma_2}$; Similarly, $\vec{\mathbf{r}}'_{0,0}$ and $\vec{\mathbf{r}}'_{1,1}$ are of length ℓ_1 and their i-th element is a random Σ_2-partial-share, i.e., $\vec{\mathbf{r}}_{0,0}[i] \leftarrow \mathsf{LRPartial}_{\Sigma_2}$ and $\vec{\mathbf{r}}_{1,1}[i] \leftarrow \mathsf{LRPartial}_{\Sigma_2}$.

For every fixed choice of $\mathbf{r}$ we define the leakage function $\nu_1^{\mathbf{r}}$ that outputs $2\mu' \leq \mu$ bits in the following way:

1. *(Input)* The function takes as an input a Σ_1-share $\mathbf{y}_1$.

2. *(Emulating R_1's view)* The function defines $\vec{\mathbf{r}}_{1,1}$ to be a length-ℓ_1 vector whose i-th element is the Σ_2-share defined by $(\vec{\mathbf{r}}'_{1,1}[i], \mathsf{LRUpdate}_{\Sigma_2}(\mathbf{y}_1[i], \vec{\mathbf{r}}_{1,0}[i], \vec{\mathbf{r}}'_{1,1}[i]))$, where $\mathbf{y}_1[i]$ is the i-th bit of $\mathbf{y}_1$. (That is, $(\vec{\mathbf{r}}_{1,0}[i], \vec{\mathbf{r}}_{1,1}[i])$ define a Σ_2-sharing of $\mathbf{y}_1[i]$.)

   We think of $\vec{\mathbf{r}}_{1,0}$ as the variable $\vec{\mathbf{y}}_{1,0}^{\mathbf{x}'_0,\mathbf{x}'_1}$ and of $\vec{\mathbf{r}}_{1,1}$ as the variable $\vec{\mathbf{y}}_{1,1}^{\mathbf{x}'_1,\mathbf{x}'_1}$. The function computes R_1's view given those values, to obtain $(\mathsf{pView}_{R_1}, \mathbf{m}_1)$.

3. *(Emulating S_1's view)* We think of $\vec{\mathbf{r}}_{0,1}$ as the variable $\vec{\mathbf{y}}_{1,0}^{\mathbf{x}'_0,\mathbf{x}'_1}$, and, as we said before, we think of $\vec{\mathbf{r}}_{1,1}$ as the variable $\vec{\mathbf{y}}_{1,1}^{\mathbf{x}'_1,\mathbf{x}'_1}$. The function computes S_1's view given those values, to obtain (pView_{S_1}).

4. *(Output)* The function outputs $\tau_1(\mathsf{pView}_{S_1}, \mathsf{pView}_{R_1}, \mathbf{m}_1)$.

Since Σ_1 is (μ, ϵ_1)-local leakage resilient, and $\mu \geq 4\mu'$, the privacy of Σ_1 guarantees that for every $\mathbf{r}$,

$$\mathbf{r}, \mathbf{s}_0^0, \nu_1^{\mathbf{r}}(\mathbf{s}_1^0) \equiv_{\epsilon_1} \mathbf{r}, \mathbf{s}_0^1, \nu_1^{\mathbf{r}}(\mathbf{s}_1^1), \tag{8}$$

where $(\mathbf{s}_0^0, \mathbf{s}_1^0) \leftarrow \mathsf{LRShare}_{\Sigma_1}(0)$ and $(\mathbf{s}_0^1, \mathbf{s}_1^1) \leftarrow \mathsf{LRShare}_{\Sigma_1}(1)$. In particular, Equation 8 holds even when $\mathbf{r}$ is uniformly distributed.

Consider the procedure that given $(\mathbf{r}, \mathbf{s}_0, \mathsf{leak})$ does the following:

1. The procedure defines $\vec{\mathbf{r}}_{0,0}$ to be a length-ℓ_1 vector whose i-th element is the Σ_2-share defined by $(\vec{\mathbf{r}}'_{0,0}[i], \mathsf{LRUpdate}(\mathbf{s}_0[i], \vec{\mathbf{r}}_{0,1}[i], \vec{\mathbf{r}}'_{0,0}[i]))$, where $\mathbf{s}_0[i]$ is the i-th bit of $\mathbf{s}_0$. (That is, $(\vec{\mathbf{r}}_{0,0}[i], \vec{\mathbf{r}}_{0,1}[i])$ define a Σ_2-sharing of $\mathbf{s}_0[i]$.)

   We think of $\vec{\mathbf{r}}_{i,j}$ as the variable $\vec{\mathbf{y}}_{i,j}^{\mathbf{x}'_0,\mathbf{x}'_1}$. The procedure computes R_0's view and S_0's view given those values, to obtain $(\mathsf{pView}_{R_0}, \mathbf{m}_0)$ and pView_{S_0}, respectively.

2. We set $\mathsf{pView}_{P_1} := (\mathsf{pView}_{S_0}, \mathsf{pView}_{R_0})$ and $\mathsf{out}_{P_1} := \mathbf{m}_0$, and we think of leak as the leakage from P_2.

3. The procedure outputs $\mathsf{pView}_{P_1}, \mathsf{leak}$.

It is not hard to see the if we apply the procedure on the LHS of Equation 8 when $\mathbf{r}$ is uniformly distributed then we obtain $\mathsf{hybrid}''_{2.1}$, and when we apply the procedure on the RHS of Equation 8 we obtain hybrid''_2. Therefore, the hybrids are ϵ_1-close, in contradiction.

Back to the general case. We have just proved that $\mathsf{hybrid}'_2 \equiv_{\epsilon_1} \mathsf{hybrid}'_{2.1}$ for the two-party case. Let us go back to the general case and prove that $\mathsf{hybrid}_2 \equiv_{\epsilon_1} \mathsf{hybrid}_{2.1}$. First, consider the four-party case, and let us assume without loss of generality that $\phi(S_0) = P_1, \phi(S_1) = P_2, \phi(R_0) = P_3$ and $\phi(R_1) = P_4$. Recall that $Z_S \subsetneq \{S_0, S_1\}$ and $Z_R \subsetneq \{R_0, R_1\}$. We may therefore assume that there is exactly one sender in Z_S and exactly one receiver in Z_R (indeed, if Z_S (resp., Z_R) is empty, then adding senders (resp., receivers) to the set may only increase the statistical distance between hybrid_2 and $\mathsf{hybrid}_{2.1}$). By symmetry, we may also assume that $Z_R = \{S_0\}$ and $Z_R = \{R_0\}$. Now, if hybrid_2 is ϵ_1-far from $\mathsf{hybrid}_{2.1}$ with respect to the leakage functions τ_{S_1}, τ_{R_1} that output μ' bits, then it is not hard to see that hybrid'_2 is ϵ_1-far from $\mathsf{hybrid}'_{2.1}$ with respect to the leakage function $\tau'_{P_2}(\mathsf{pView}_{S_1}, \mathsf{pView}_{R_1}, \mathbf{m}_1) := (\tau_{S_1}(\mathsf{pView}_{S_1}), \tau_{R_1}(\mathsf{pView}_{R_1}, \mathbf{m}_1))$ that outputs $2\mu'$ bits, in

contradiction. A similar argument also applies to the three-party case. This concludes the proof of the claim. □

Proof of Claim B.8. We first observe that the step of "refreshing shares" in both hybrids is executed in the same way, and therefore the random variables $(\mathsf{pView}_0^{\mathsf{refresh}}, \mathbf{x}'_0)$ and $(\mathsf{pView}_1^{\mathsf{refresh}}, \mathbf{x}'_1)$ have the same distribution in both hybrids. Let us fix those random variables. Similarly, in the step of "generating matrix of evaluations" the random variables $(\mathsf{pView}_0^{\mathsf{matrix}}, \mathsf{pView}_1^{\mathsf{matrix}})$ and $(\vec{\mathbf{y}}_{0,0}^{\mathbf{x}'_0,\mathbf{x}'_1}, \vec{\mathbf{y}}_{0,1}^{\mathbf{x}'_0,\mathbf{x}'_1}, \vec{\mathbf{y}}_{1,0}^{\mathbf{x}'_0,\mathbf{x}'_1}, \vec{\mathbf{y}}_{1,1}^{\mathbf{x}'_0,\mathbf{x}'_1})$ are sampled in the same way in both distributions, and we fix them as well. In the steps of transmitting rows and columns (Steps (3)–(4)), the random variables $(\vec{\mathbf{y}}_{0,0}^{\mathbf{x}'_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{0,1}^{\mathbf{x}'_0,\mathbf{z}_1})_{\mathbf{z}_1\in\{0,1\}^{\ell_1}}$ and $(\vec{\mathbf{y}}_{1,0}^{\mathbf{z}_0,\mathbf{x}'_1}, \vec{\mathbf{y}}_{1,1}^{\mathbf{z}_0,\mathbf{x}'_1})_{\mathbf{z}_0\in\{0,1\}^{\ell_1}}$ have the same distribution in both hybrids, and we fix them as well. Finally, in the steps "computing the output" and in sampling the outputs, the random variables $(\mathsf{pView}_0^{\mathsf{out}}, \mathbf{w}_0)$, $(\mathsf{pView}_1^{\mathsf{out}}, \mathbf{w}_1)$, $\mathbf{m}_0$ and $\mathbf{m}_1$ have the same distribution and we fix them as well. This means that the only random variables that are not fixed are

$$((\vec{\mathbf{y}}_{0,0}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{1,0}^{\mathbf{z}_0,\mathbf{z}_1}), (\vec{\mathbf{y}}_{0,1}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{1,1}^{\mathbf{z}_0,\mathbf{z}_1}))_{(\mathbf{z}_0,\mathbf{z}_1):\mathbf{z}_0\neq\mathbf{x}'_0\wedge\mathbf{z}_1\neq\mathbf{x}'_1},$$

and we note that they are independent from the values we've fixed so far. In other words, S_0 and S_1 hold Σ_2-sharings of (less than) $2\cdot 2^{2\ell_1}\cdot\ell_1$ bits. The claim now follows from Claim B.1. □

Proof of Claim B.9. Assume towards contradiction that the claim is not true, and that $\mathsf{hybrid}_{2.2}$ is $\mathsf{size}(\mathsf{matrix})\cdot\epsilon_2$-far from $\mathsf{hybrid}_{2.3}$. We observe that the random variables $(\mathsf{pView}_0^{\mathsf{refresh}}, \mathbf{x}'_0)$ and $(\mathsf{pView}_1^{\mathsf{refresh}}, \mathbf{x}'_1)$ in the step of "refreshing shares", the random variables $(\vec{\mathbf{y}}_{0,0}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{0,1}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{1,0}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{1,1}^{\mathbf{z}_0,\mathbf{z}_1})_{\mathbf{z}_0,\mathbf{z}_1\in\{0,1\}^{\ell_1}}$, and the random variables $(\mathsf{pView}_0^{\mathsf{out}}, \mathbf{w}_0)$, $(\mathsf{pView}_1^{\mathsf{out}}, \mathbf{w}_1)$, $\mathbf{m}_0$ and $\mathbf{m}_1$ in the steps "computing the output" and the output sampling, have the same distribution in both hybrids, and therefore there is a fixed choice of them such that $\mathsf{hybrid}_{2.2}$ is $\mathsf{size}(\mathsf{matrix})\cdot\epsilon_2$-far from $\mathsf{hybrid}_{2.3}$ even conditioned on those variables. Let us denote the conditional hybrids by $\mathsf{hybrid}'_{2.2}$ and $\mathsf{hybrid}'_{2.3}$.

We observe that the only random variables that were not fixed are $(\mathsf{pView}_0^{\mathsf{matrix}}, \mathsf{pView}_1^{\mathsf{matrix}})$, and that they are independent of the random variables that we've fixed. We further observe that the partial views of R_0 and R_1 are fixed, and given $\mathsf{pView}_i^{\mathsf{matrix}}$ we can compute the partial view of S_i. Let τ'_i be the leakage function that takes $(\mathsf{pView}_i^{\mathsf{matrix}}, \mathsf{out}_i^{\mathsf{matrix}})$, ignores the output $\mathsf{out}_i^{\mathsf{matrix}}$, uses the values that we fixed to compute the partial view of S_i, denoted pView_{S_i}, as well as the fixed partial views $(\mathsf{pView}_{R_j})_{R_j\in\phi^{-1}(\phi(S_i))}$, sets $\mathsf{pView}_{\phi(S_i)} := (\mathsf{pView}_{S_i}, (\mathsf{pView}_{R_j})_{R_j\in\phi^{-1}(\phi(S_i))})$ and outputs $\tau_{\phi(S_i)}(\mathsf{pView}_{\phi(S_i)})$. (Here we use the fact that $\phi(S_0)\neq\phi(S_1)$.) Then it is not hard to verify that the privacy of the $(\mu, \mathsf{size}(\mathsf{matrix})\cdot\epsilon_2)$-composable leakage-resilient passive security of $\mathsf{LR2PC}_{\Sigma_2}(f_{\mathsf{matrix}})$ (see Theorem 4.11) is violated with respect to the inputs $(\mathbf{a}_0, \mathbf{b}_0)$ and $(\mathbf{a}_1, \mathbf{b}_1)$, the set Z_S and the leakage functions τ'_0, τ'_1. This concludes the proof of the claim. □

Proof of Claim B.10. Assume towards contradiction that the claim is not true, and that $\mathsf{hybrid}_{2.3}$ is $\mathsf{size}(\mathsf{matrix})\cdot\epsilon_2$-far from hybrid_3. We observe that the random variables $(\mathsf{pView}_0^{\mathsf{refresh}}, \mathbf{x}'_0)$ and $(\mathsf{pView}_1^{\mathsf{refresh}}, \mathbf{x}'_1)$ in the step of "refreshing shares", the random variables $(\vec{\mathbf{y}}_{0,0}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{0,1}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{1,0}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{1,1}^{\mathbf{z}_0,\mathbf{z}_1})_{\mathbf{z}_0,\mathbf{z}_1\in\{0,1\}^{\ell_1}}$, and the random variables $(\mathsf{pView}_0^{\mathsf{out}}, \mathbf{w}_0)$, $(\mathsf{pView}_1^{\mathsf{out}}, \mathbf{w}_1)$, $\mathbf{m}_0$ and $\mathbf{m}_1$ in the steps "computing the output" and output sampling have the same distribution in both hybrids, and therefore there is a fixed choice of them such that $\mathsf{hybrid}_{2.3}$ is $\mathsf{size}(\mathsf{matrix})\cdot\epsilon_2$-far from hybrid_3 even conditioned on those variables. Let us denote the conditional hybrids by $\mathsf{hybrid}'_{2.3}$ and hybrid'_3.

We observe that the only random variables that were not fixed are $(\mathsf{pView}_0^{\mathsf{matrix}}, \mathsf{pView}_1^{\mathsf{matrix}})$. In $\mathsf{hybrid}'_{2.3}$ these random variables are sampled in an execution of $\mathsf{LR2PC}_{\Sigma_2}(f_{\mathsf{matrix}})$ with inputs $(\mathbf{a}_0, \mathbf{b}_0)$ to S_0, and $(\mathbf{a}_1, \mathbf{b}_1)$ to S_1, while in hybrid_3 these random variables are sampled in an execution of $\mathsf{LR2PC}_{\Sigma_2}(f_{\mathsf{matrix}})$ with inputs $((\mathbf{a}_0, \mathbf{b}_0), (\mathbf{a}_1, \mathbf{b}_1))$ conditioned on the fixed outputs $(\vec{\mathbf{y}}_{0,0}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{0,1}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{1,0}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{1,1}^{\mathbf{z}_0,\mathbf{z}_1})_{\mathbf{z}_0,\mathbf{z}_1 \in \{0,1\}^{\ell_1}}$. We further observe that the partial views of R_0 and R_1 are fixed, and given $\mathsf{pView}_i^{\mathsf{matrix}}$ we can compute the partial view of S_i. Let τ'_i be the leakage function that takes $\mathsf{pView}_i^{\mathsf{matrix}}$, uses the values that we fixed to compute the partial view of S_i, denoted pView_{S_i}, as well as the fixed partial views $(\mathsf{pView}_{R_j})_{R_j \in \phi^{-1}(\phi(S_i))}$, sets $\mathsf{pView}_{\phi(S_i)} := (\mathsf{pView}_{S_i}, (\mathsf{pView}_{R_j})_{R_j \in \phi^{-1}(\phi(S_i))})$ and outputs $\tau_{\phi(S_i)}(\mathsf{pView}_{\phi(S_i)})$. Then it is not hard to verify that the real-world output independence of the strong $(\mu, \mathsf{size}(\mathsf{matrix}) \cdot \epsilon_2)$-composable leakage-resilient passive security of $\mathsf{LR2PC}_{\Sigma_2}(f_{\mathsf{matrix}})$ is violated with respect to the inputs $(\mathbf{a}_0, \mathbf{b}_0)$ and $(\mathbf{a}_1, \mathbf{b}_1)$, the set Z_S and the leakage functions τ'_0, τ'_1. This concludes the proof of the claim. □

B.2.5 Proof of Claim B.5

Let $Z_R = Z' \cap \{R_0, R_1\}$. It is not hard to see that if $Z' = \{R_0, R_1\}$ then $\mathsf{hybrid}_4 \equiv \mathsf{hybrid}_5$. Therefore, let us assume that $Z_R \subsetneq \{R_0, R_1\}$. Assume towards contradiction that hybrid_4 is $\mathsf{size}(\mathsf{out}) \cdot \epsilon_2$-far from hybrid_5. We observe that the random variables $(\mathsf{pView}_0^{\mathsf{refresh}}, \mathbf{x}'_0)$ and $(\mathsf{pView}_1^{\mathsf{refresh}}, \mathbf{x}'_1)$ in the step of "refreshing shares", the random variables $(\mathsf{pView}_0^{\mathsf{matrix}}, \mathsf{pView}_1^{\mathsf{matrix}}), (\vec{\mathbf{y}}_{0,0}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{0,1}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{1,0}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{1,1}^{\mathbf{z}_0,\mathbf{z}_1})_{\mathbf{z}_0,\mathbf{z}_1 \in \{0,1\}^{\ell_1}}$ from the step "generating matrix of evaluations", and the random variables $\mathbf{m}_0$ and $\mathbf{m}_1$ in the step "translating the shares" have the same distribution in both hybrids, and therefore there is a fixed choice of them such that hybrid_4 is $\mathsf{size}(\mathsf{out}) \cdot \epsilon_2$-far from hybrid_5 even conditioned on these fixed values. Denote the conditioned hybrids by hybrid'_4 and hybrid'_5.

We observe that the only random variables that were not fixed are $\mathsf{pView}_0^{\mathsf{out}}, \mathsf{pView}_1^{\mathsf{out}}, \mathbf{w}_0, \mathbf{w}_1$, and that in both hybrids they are independent of the random variables that we've fixed. We also observe that the views of S_0 and S_1 are fixed, and we denote them by pView_{S_0} and pView_{S_1}, respectively. In addition, given $\mathsf{pView}_i^{\mathsf{out}}$ and $\mathbf{w}_i$ we can recover the partial view and the output of R_i, denoted $(\mathsf{pView}_{R_i}, \mathbf{m}_i)$, using the values that we've fixed. Let τ'_i be the leakage function that takes $(\mathsf{pView}_i^{\mathsf{out}}, \mathsf{out}_i^{\mathsf{out}})$, computes the partial view of R_i and its output, denoted $(\mathsf{pView}_{R_i}, \mathbf{m}_i)$, computes the partial view of every sender $S_j \in \phi^{-1}(\phi(R_i))$, denoted pView_{S_j}, and outputs $\tau_{\phi(R_i)}(\mathsf{pView}_{R_i}, \mathbf{m}_i, (\mathsf{pView}_{S_j})_{S_j \in \phi^{-1}(\phi(R_i))})$. Then it is not hard to verify that the strong $(\mu, \mathsf{size}(\mathsf{out}) \cdot \epsilon_2)$-composable leakage-resilient passive security of $\mathsf{LR2PC}(f_{\mathsf{out}})$ is violated with respect to the (fixed) inputs $(\mathbf{x}'_0, (\mathbf{y}_0^{\mathbf{x}'_0,\mathbf{z}_1})_{\mathbf{z}_1 \in \{0,1\}^{\ell_1}})$ and $(\mathbf{x}'_1, (\mathbf{y}_1^{\mathbf{z}_0,\mathbf{x}'_1})_{\mathbf{z}_0 \in \{0,1\}^{\ell_1}})$, the set Z_R, the leakage functions τ'_0, τ'_1, and the output-string m. This concludes the proof of the claim.

B.2.6 Proof of Claim B.6

Let $Z_R = Z' \cap \{R_0, R_1\}$. It is not hard to see that if $Z_R = \{R_0, R_1\}$ then $\mathsf{hybrid}_5 \equiv \mathsf{hybrid}_6$. Indeed, in this case in Hybrid 5 it holds that $m = a \cdot x \oplus b$ and $(\mathbf{m}_0, \mathbf{m}_1) \leftarrow \mathsf{LRShare}_{\Sigma_1}(m)$. Therefore, let us assume that $Z_R \subsetneq \{R_0, R_1\}$. Assume towards contradiction that hybrid_5 is ϵ_1-far from hybrid_6. We observe that the random variables $(\mathsf{pView}_0^{\mathsf{refresh}}, \mathbf{x}'_0)$ and $(\mathsf{pView}_1^{\mathsf{refresh}}, \mathbf{x}'_1)$ in the step of "refreshing shares", the random variables $(\mathsf{pView}_0^{\mathsf{matrix}}, \mathsf{pView}_1^{\mathsf{matrix}}), (\vec{\mathbf{y}}_{0,0}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{0,1}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{1,0}^{\mathbf{z}_0,\mathbf{z}_1}, \vec{\mathbf{y}}_{1,1}^{\mathbf{z}_0,\mathbf{z}_1})_{\mathbf{z}_0,\mathbf{z}_1 \in \{0,1\}^{\ell_1}}$ in the step "generating matrix of evaluations", and the random variables $(\mathsf{pView}_0^{\mathsf{out}}, \mathbf{w}_0)$, $(\mathsf{pView}_1^{\mathsf{out}}, \mathbf{w}_1)$ in the step "computing the output" have the same distribution in both hybrids,

and therefore there is a fixed choice of them such that hybrid_5 is ϵ_1-far from hybrid_6 even conditioned on these fixed values. Denote the conditioned hybrids by hybrid'_5 and hybrid'_6. We observe that the only random variables that were not fixed are $\mathbf{m}_0, \mathbf{m}_1$. The claim now follows by the privacy of Σ_1. This concludes the proof of the claim.

B.3 Proof of Lemma 5.7

Fix an input $\mathbf{x} \in \{0,1\}^\ell$, a set $Z \subseteq \{p_1, \ldots p_n\}$ with $\chi(Z) = 0$, and leakage functions $\tau_1, \ldots, \tau_n$ that output $\mu = \lfloor \log(\kappa) \rfloor$ bits. Let us observe that since χ is admissible then for every w_i it holds that $p_{\mathsf{Left}(w_i)} \neq p_{\mathsf{Right}(w_i)}$. We begin with perfect correctness.

Perfect correctness. Perfect correctness follows by a standadrd inductive argument, proving that for every execution E of (π, Enc) on input $\mathbf{x}$, there is an execution E^h of π_h on inputs $(\mathbf{x}^1, \ldots, \mathbf{x}^m)$ that satisfy $\mathbf{x}^1 \oplus \ldots \oplus \mathbf{x}^m = \mathbf{x}$, such that for every variable x^k in π_h it holds that $X^{k,0}$ and $X^{k,1}$ store a Σ_1-sharing of the the value of x^k. This claim follows by the perfect correctness of $\mathsf{LROp}(*)$ and LROLE (Lemma 5.1 and Lemma 5.2), and since we compile a randomness statement by picking random Σ_1-shares, that form a Σ_1-sharing of a random variables. This now implies that the abort flag is never raised, and that the outputs of the parties is indeed $f(\mathbf{x})$.

We continue by analysing statistical privacy.

B.3.1 Statistical Privacy

To prove that our protocol satisfies statistical privacy, we define the following hybrids.

hybrid_0. In hybrid_0 the views $\mathsf{view}_1, \ldots, \mathsf{view}_n$ are generated by $\mathsf{Sim}(Z, f(\mathbf{x}))$, and the output is set to be $(\mathsf{view}_i)_{p_i \in Z}, (\tau_i(\mathsf{view}_i))_{p_i \notin Z}$.

$\mathsf{hybrid}_{0.1}$. We set $\mathsf{hybrid}_{0.1}$ to be identical to hybrid_0, except that in the fake input initialization stage of Sim, the inputs are sampled like in the input encoder, i.e., we sample an m-out-of-m secret sharing of $\mathbf{x}$, denoted $(\mathbf{x}^1, \ldots, \mathbf{x}^m)$, and for every $i \in [m]$ we compute $(\vec{\mathbf{x}}_0^i, \vec{\mathbf{x}}_1^i) \leftarrow \mathsf{LRvShare}_{\Sigma_1}(\mathbf{x}^i)$.

Recall that we assume without loss of generality that all input statements appear first as a prefix of the protocol, and every input bit is read at most once. Let us denote by i^* the number of input statements in π_h.

hybrid_i for $i \geq i^*$. For every $i^* \leq i \leq |\pi_h|$ we define hybrid_i that, at a high level, executes the compilation of the first i statements of π_h like in π, and the compilation of the remaining statements is executed like in Sim. Formally, we define the hybrid in the following way.

hybrid_i

In the hybrid, we first generate the views $\mathcal{V} = (\mathsf{view}_1, \ldots, \mathsf{view}_n)$ of the parties, and then output $(\mathsf{view}_j)_{p_j \in Z}, (\tau_j(\mathsf{view}_j))_{p_j \in Z}$. We continue by explaining how to generate the views.

Sampling inputs. First, we generate real-world inputs by executing the input encoder, i.e., we sample an m-out-of-m secret sharing of $\mathbf{x}$, denoted $(\mathbf{x}^1, \ldots, \mathbf{x}^m)$, and for every $i \in [m]$ we compute $(\vec{\mathbf{x}}_0^i, \vec{\mathbf{x}}_1^i) \leftarrow \mathsf{LRvShare}_{\Sigma_1}(\mathbf{x}^i)$. For every $i \in [n]$ set $\mathsf{input}_i := (\mathbf{x}_0^j)_{\mathsf{Left}(w_j)=i} \cup (\mathbf{x}_1^j)_{\mathsf{Right}(w_j)=i}$.

Views for π_h. We execute the protocol π_h with $w_1, \ldots, w_m$ on inputs $(\mathbf{x}^1, \ldots, \mathbf{x}^m)$ to obtain the views $\mathcal{V}^h := (\mathsf{view}_1^h, \ldots, \mathsf{view}_m^h)$.

Initializing $\mathcal{V}$. We initialize empty views $\mathcal{V} = (\mathsf{view}_1, \ldots, \mathsf{view}_n)$. At a high level, we update $\mathcal{V}$ by iterating over the instructions d_j of π_h for all $j = 1, \ldots, |\pi_h|$. For every $j \leq i$ we update $\mathcal{V}$ like in a real-world execution of π on inputs $(\mathsf{input}_1, \ldots, \mathsf{input}_n)$ conditioned on the execution being consistent with $\mathcal{V}^h$, i.e., for every variable x^k in π_h that is defined in the first i statements it holds that the variables $X^{k,0}$ and $X^{k,1}$ store a Σ_1-sharing of the value of x^k according to $\mathcal{V}^h$. For every $j \geq i$, the instructions are updated according to the simulator Sim, with respect to the views $\mathcal{V}' := (\mathsf{view}_i^h)_{w_i \in Z_h}$. In the following, we denote the state of the views $\mathcal{V}$ *after* the update of d_j by $\mathcal{V}_j$.

First i steps: Real-world execution. and update the views as follows.

1. The first i^* steps, that correspond to reading the inputs, are executed like in π with respect to the inputs $\mathsf{input}_1, \ldots, \mathsf{input}_n$.

   We observe that $\mathcal{V}^h$ was generated by executing π_h on the inputs $(\mathbf{x}^1, \ldots, \mathbf{x}^m)$, and that $(\vec{\mathbf{x}}_0^i[j], \vec{\mathbf{x}}_1^i[j])$ is a fresh Σ_1-sharing of $\mathbf{x}^i[j]$. Therefore, for every variable x^k that is defined in the first i^* steps of π_h, the lists of variables $X^{k,0}$ and $X^{k,1}$ in π store a fresh Σ_1-sharing of the value of x^k according to $\mathcal{V}^h$.

2. For every $j = i^* + 1, \ldots, i$, and for the j-th statement d_j in π_h, we update $\mathcal{V}$ by executing the compilation of the j-th statement of π_h in the following way:

   - If the j-th statement is $\mathsf{comp}(w_j, \mathsf{rand}, \emptyset, x^k)$ then we let v be the value of x^k according to $\mathcal{V}^h$, we sample $(\mathbf{v}_0, \mathbf{v}_1) \leftarrow \mathsf{LRShare}_{\Sigma_1}(v)$ and store $\mathbf{v}_0$ in $X^{k,0}$ and $\mathbf{v}_1$ in $X^{k,1}$.
   - The compilation of every other statement is executed like in π with respect to the current state of the views $\mathcal{V}_{j-1}$.

   Observe that for every variable x^k that is defined in the first i steps of π_h, the lists of variables $X^{k,0}$ and $X^{k,1}$ in π store a fresh Σ_1-sharing of the value of x^k according to $\mathcal{V}^h$. Indeed, any x^k that is not defined as part of an input or randomness statement, is fully defined by all previously-set variables.

Steps $i+1, \ldots, |\pi_h|$: Simulation. Define $\mathcal{V}' := (\mathsf{view}_i^h)_{w_i \in Z_h}$. For every $j = i+1, \ldots, |\pi_h|$, and for the j-th statement d_j in π_h, we update the views $\mathcal{V}$ in the following way: the compilation of the j-th statement d_j of π_h is executed like in the "protocol simulation" stage of Sim, with respect to the inputs $(\mathsf{input}_1, \ldots, \mathsf{input}_n)$ of the parties, the π_h-simulated views $\mathcal{V}'$, and the views $\mathcal{V}_{j-1}$.

Figure 21: hybrid_i

Observe the in hybrid_0 the views are generated like in the simulation. In addition, it is not hard to verify that in $\mathsf{hybrid}_{|\pi|}$ the views are generated like in a real-world execution. We also note that, by the perfect privacy of π_h, it holds that $\mathsf{hybrid}_{i^*} \equiv \mathsf{hybrid}_{0.1}$. Therefore, it suffices to prove the following claims.

Claim B.11. $\mathsf{SD}(\mathsf{hybrid}_0, \mathsf{hybrid}_{0.1}) \leq \frac{i^*}{\kappa \cdot |\pi_h|}$.

Claim B.12. *For every $i \in \{i^* + 1, \ldots, |\pi_h|\}$ it holds that* $\mathsf{SD}(\mathsf{hybrid}_{i-1}, \mathsf{hybrid}_i) \leq \frac{1}{\kappa \cdot |\pi_h|}$.

We continue with the proofs of the two claims.

Proof of Claim B.11. First, let us observe that $Z_h \subsetneq \{w_1, \ldots, w_m\}$. Indeed, for every $Z' \subseteq \{p_1, \ldots, p_n\}$ and $Z'_h := \left\{w_i : p_{\mathsf{Left}(w_i)}, p_{\mathsf{Right}(w_i)} \in Z'\right\}$ it holds that $\chi(Z') = h(Z'_h)$ since $\chi = (h, \mathsf{AND}_2^m, T)$. In addition, χ is an admissible monotone formula, so $1 = \chi(\{p_1, \ldots, p_n\}) = h(\{w_1, \ldots, w_m\})$. However, in our case $0 = \chi(Z) = h(Z_h)$, and therefore $Z_h \subsetneq \{w_1, \ldots, w_m\}$. To simplify notation, let us assume without loss of generality that $w_1 \notin Z_h$, and that $\mathsf{Left}(w_1) = 1$ and $\mathsf{Right}(w_1) = 2$.

Assume towards contradiction that hybrid_0 is $i^*/(\kappa \cdot |\pi_h|)$-far from $\mathsf{hybrid}_{0.1}$. Observe that both in hybrid_0 and in $\mathsf{hybrid}_{0.1}$ the random variables $(\mathbf{x}^i, \vec{\mathbf{x}}_0^i, \vec{\mathbf{x}}_1^i)_{i=2,\ldots,n}$ have the same distribution, and therefore there is a fixed choice of those random variables, such that hybrid_0 is $i^*/(\kappa \cdot |\pi_h|)$-far from $\mathsf{hybrid}_{0.1}$ even conditioned on those random variables. Denote the hybrids conditioned on those values by hybrid'_0 and $\mathsf{hybrid}'_{0.1}$. Observe that in hybrid'_0 the random variable $\mathbf{x}^1$ is uniformly distributed, while in $\mathsf{hybrid}'_{0.1}$ the random variable $\mathbf{x}^1$ is fixed to $\mathbf{x}_1 = \mathbf{x} \oplus \mathbf{x}^2 \oplus \ldots \oplus \mathbf{x}^m$. In both hybrids we have $(\vec{\mathbf{x}}_0^1, \vec{\mathbf{x}}_1^1) \leftarrow \mathsf{LRvShare}_{\Sigma_1}(\mathbf{x}^1)$. To distinguish between $(\mathbf{x}^1, \vec{\mathbf{x}}_0^1, \vec{\mathbf{x}}_1^1)$ in hybrid'_0 and in $\mathsf{hybrid}'_{0.1}$, we denote the corresponding random variables in $\mathsf{hybrid}'_{0.1}$ by $(\mathbf{z}^1, \vec{\mathbf{z}}_0^1, \vec{\mathbf{z}}_1^1)$.

We continue by showing that we can distinguish the hybrid'_0 random variables $(\vec{\mathbf{x}}_0^1, \vec{\mathbf{x}}_1^1)$ from $\mathsf{hybrid}_{0.1}$ random variables $(\vec{\mathbf{z}}_0^1, \vec{\mathbf{z}}_1^1)$, in contradiction to Claim B.1. To do so, We first observe that, since $w_1 \notin Z_h$, the simulation of every statement that is not of the form $\mathsf{input}(w_1, j, x^k)$ does not depend on the random variables $(\vec{\mathbf{x}}_0^1, \vec{\mathbf{x}}_1^1)$. Therefore, there is a fixed choice of $\mathsf{view}_3^*, \ldots, \mathsf{view}_n^*$, together with "partial views" $\widetilde{\mathsf{view}}_1$ and $\widetilde{\mathsf{view}}_2$ that are only missing the Σ_1-shares of $\mathbf{x}^1$ (i.e., they are only missing the values of variables $X^{k,0}$ and $X^{k,1}$ for variables x^k that are defined in input statements of the form $\mathsf{input}(w_1, j, x^k)$) such that hybrid'_0 is $i^*/(\kappa \cdot |\pi_h|)$-far from $\mathsf{hybrid}'_{0.1}$ even conditioned on those views. Denote the hybrids conditioned on those values by hybrid''_0 and $\mathsf{hybrid}''_{0.1}$.

Define the leakage function ν_0 (resp., ν_1) in the following way:

1. The function takes as an input $\vec{\mathbf{x}}_0^1$ that corresponds to p_1 (resp., $\vec{\mathbf{x}}_1^1$ that corresponds to p_2).
2. Given $\vec{\mathbf{x}}_0^1$ (resp., $\vec{\mathbf{x}}_1^1$) and the fixed view $\widetilde{\mathsf{view}}_1$ (resp., $\widetilde{\mathsf{view}}_2$) that is only missing $\vec{\mathbf{x}}_0^1$ (resp., $\vec{\mathbf{x}}_1^1$), the function computes the complete view $\mathsf{view}_1^* = (\mathbf{x}_0^1, \widetilde{\mathsf{view}}_1)$ (resp., $\mathsf{view}_2^* = (\mathbf{x}_1^1, \widetilde{\mathsf{view}}_2)$).
3. The function outputs $\tau_1(\mathsf{view}_1^*)$ (resp., $\tau_2(\mathsf{view}_2^*)$).

Let us assume without loss of generality that $p_1 \notin Z$ and $p_2 \in Z$. Recall that Σ_1 is (μ_1, ϵ_1)-local leakage resilient with $\mu_1 \geq \mu$ and $\epsilon_1 \leq 1/(\kappa\,|\pi_h|)$. Since every input bit is read exactly once, the length of $(\mathbf{x}^i)$ is at most i^*, and therefore, it holds that

$$\nu_0(\vec{\mathbf{x}}_0^1),\ \vec{\mathbf{x}}_1^1 \equiv_{i^*/(\kappa|\pi_h|)}\ \nu_0(\vec{\mathbf{z}}_0^1),\ \vec{\mathbf{z}}_1^1,$$

In particular, it holds that

$$\tau_1(\mathbf{x}_0^1, \widetilde{\mathsf{view}}_1), (\mathbf{x}_1^1, \widetilde{\mathsf{view}}_2), \mathsf{view}_3^*, \ldots, \mathsf{view}_n^* \equiv_{i^*/(\kappa|\pi_h|)} \tau_1(\mathbf{z}_0^1, \widetilde{\mathsf{view}}_1), (\mathbf{z}_1^1, \widetilde{\mathsf{view}}_2), \mathsf{view}_3^*, \ldots, \mathsf{view}_n^*.$$

Since the LHS corresponds to the views in hybrid''_0 and the RHS corresponds to the views in $\mathsf{hybrid}''_{0.1}$ it means that hybrid''_0 is $i^*/(\kappa\,|\pi_h|)$-close to $\mathsf{hybrid}''_{0.1}$. This concludes the proof of the claim. □

Proof of Claim B.12. Assume towards contradiction that hybrid_{i-1} is $1/(\kappa \cdot |\pi_h|)$-far from hybrid_i. First, we observe that $\mathcal{V}^h$ has the same distributions in both hybrids, and that conditioned on $\mathcal{V}^h$, the random variable $\mathcal{V}_{i-1}$ has the same distribution in both hybrids. Therefore there is a fixed

choice of $(\mathcal{V}^h, \mathcal{V}_{i-1})$ such that hybrid_{i-1} is $1/(\kappa \cdot |\pi_h|)$-far from hybrid_i even conditioned on $(\mathcal{V}^h, \mathcal{V}_{i-1})$. Let us denote the conditional hybrids by hybrid'_{i-1} and hybrid'_i.

We split into cases according to the i-th statement. We observe that all input statements appear before the i-th statement, and therefore we only need to consider the following statements.

Output and abort statements. If the statement is $\mathsf{out}(w, j, x^k)$ or $\mathsf{abort}(w_j, x^k)$ then $\mathsf{hybrid}_{i-1} \equiv \mathsf{hybrid}_i$. Indeed, the compilation of the i-th statement is performed in the same way in both hybrids: for an output statement, in both hybrids the output of $\mathcal{F}^{\mathsf{out}}_{\Sigma_1}$ is y_j and the message from $p_{\mathsf{Right}(w)}$ to $p_{\mathsf{Left}(w)}$ is y_j, and for an abort statement, in both hybrids the output of $\mathcal{F}^{\mathsf{out}}_{\Sigma_1}$ is 0. In addition, the compilation of statements $i+1, \ldots, |\pi_h|$ in Sim is independent of the values generated in the compilation of the i-th statement.

Local randomness, addition and multiplication. Assume that the i-th statement is either (1) $\mathsf{comp}(w_j, *, (x^{k_1}, x^{k_2}), x^{k_3})$ for $* \in \{+, \times\}$, or (2) $\mathsf{comp}(w_j, \mathsf{rand}, \emptyset, x^k)$. First, we note that if $w_j \in Z_h$ then $\mathsf{hybrid}'_{i-1} \equiv \mathsf{hybrid}'_i$. Indeed, in both cases the execution of the i-th statement is performed in the same way, and the execution of statements $i+1, \ldots, |\pi_h|$ is performed in the same way.

Therefore, let us therefore assume that $w_j \notin Z_h$. Then the compilation of statements $i+1, \ldots, |\pi_h|$ in Sim is independent of the values generated in the compilation of the i-th statement. In particular, there is a fixed choice of all the variables in statements $i+1, \ldots, |\pi_h|$, such that hybrid'_{i-1} is $1/(\kappa \cdot |\pi_h|)$-far from hybrid_i even conditioned on those variables. Let us denote the conditional hybrids by hybrid''_{i-1} and hybrid''_i, and let us observe that the views $(\mathsf{view}_k)_{k \notin \{\mathsf{Left}(w_j), \mathsf{Right}(w_j)\}}$ are fixed, and that the views $\mathsf{view}_{\mathsf{Left}(w_j)}$ and $\mathsf{view}_{\mathsf{Right}(w_j)}$ are only missing the partial views and outputs of $\mathsf{LROp}_{\Sigma_1}(*)$ (in Case (1)) or $\mathsf{LRRand}_{\Sigma_1}$ (in Case (2)) from the compilation of the i-th statement. We split into cases.

- Let us first consider the case of a randomness statement $\mathsf{comp}(w_j, \mathsf{rand}, \emptyset, x^k)$. We observe that in hybrid''_{i-1} the compilation of the i-th statement of π_h is done by storing fresh Σ_1-shares of zero in $X^{k,0}$ and $X^{k,1}$, and that in hybrid''_i the compilation of the i-th statement of π_h is done by storing fresh Σ_1-shares of v in $X^{k,0}$ and $X^{k,1}$, where v is the value of x^k according to $\mathcal{V}^h$. Observe that v has to be 1 or otherwise the hybrids are identical.

  Consider the leakage function ν_0 (resp., ν_1) that given a Σ_1-share $\mathbf{x}_0$ (resp., $\mathbf{x}_1$) uses the values that we've fixed and $\mathbf{x}_0$ (resp., $\mathbf{x}_1$) to compute the view $\mathsf{view}_{\mathsf{Left}(w_j)}$ (resp., $\mathsf{view}_{\mathsf{Right}(w_j)}$), and outputs $\tau_{\mathsf{Left}(w_j)}(\mathsf{view}_{\mathsf{Left}(w_j)})$ (resp., $\tau_{\mathsf{Right}(w_j)}(\mathsf{view}_{\mathsf{Right}(w_j)})$). It is not hard to verify that the (μ_1, ϵ_1)-local leakage resilient of Σ_1, with $\mu_1 \geq \lfloor \log(\kappa) \rfloor$ and $\epsilon_1 \leq 1/(\kappa\,|\pi_h|)$, is violated with respect to the leakage functions ν_0, ν_1, and the set $Z \cap \left\{p_{\mathsf{Left}(w_j)}, p_{\mathsf{Right}(w_j)}\right\}$.

- We continue with the case of an $\mathsf{comp}(w_j, *, (x^{k_1}, x^{k_2}), x^{k_3})$ for $* \in \{+, \times\}$ statement. Let us denote the fixed inputs of $p_{\mathsf{Left}(w_j)}$ and $p_{\mathsf{Right}(w_j)}$ in the execution of $\mathsf{LROp}_{\Sigma_1}(*)$ by $(\mathbf{a}_A, \mathbf{b}_A, \mathbf{c}_A)$ and $(\mathbf{a}_B, \mathbf{b}_B, \mathbf{c}'_B)$. Let us also observe that in hybrid''_i the compilation of the i-th statement of π_h corresponds to an execution of $\mathsf{LROp}_{\Sigma_1}(*)$ with those inputs, and that in hybrid''_{i-1} the execution is simulated by sampling the outputs $(\mathsf{out}_0, \mathsf{out}_1) \leftarrow \mathsf{LRShare}_{\Sigma_1}(0)$ and obtaining the partial views from $(\mathsf{Sim}_{\mathsf{LROp}(*)}(Z \cap \left\{p_{\mathsf{Left}(w_j)}, p_{\mathsf{Right}(w_j)}\right\}, (\mathsf{out}_0, \mathsf{out}_1), \emptyset))$. We continue by showing that we can violate the $(\mu, \frac{1}{\kappa \cdot |\pi_h|})$-composable leakage-resilient passive security of

$\mathsf{LROp}_{\Sigma_1}(*)$, in contradiction to Lemma 5.1. Indeed, define the leakage function ν_0 (resp., ν_1) in the following way:

1. The function takes as an input the partial view and output $(\mathsf{pView}_0^{\mathsf{LROp}}, \mathsf{out}_0)$ (resp., $(\mathsf{pView}_1^{\mathsf{LROp}}, \mathsf{out}_1)$).
2. Given the values the we've fixed and $(\mathsf{pView}_0^{\mathsf{LROp}}, \mathsf{out}_0)$ (resp., $(\mathsf{pView}_1^{\mathsf{LROp}}, \mathsf{out}_1)$), the function compute the view $\mathsf{view}_{\mathsf{Left}(w_j)}$ (resp., $\mathsf{view}_{\mathsf{Right}(w_j)}$).
3. The function outputs $\tau_{\mathsf{Left}(w_j)}(\mathsf{view}_{\mathsf{Left}(w_j)})$ (resp., $\tau_{\mathsf{Right}(w_j)}(\mathsf{view}_{\mathsf{Right}(w_j)})$).

It is not hard to verify that the composable leakage-resilient passive security of $\mathsf{LROp}_{\Sigma_1}(*)$ is violated with respect to the inputs $(\mathbf{a}_A, \mathbf{b}_A, \mathbf{c}_A)$ and $(\mathbf{a}_B, \mathbf{b}_B, \mathbf{c}'_B)$, the leakage functions ν_0, ν_1, the set $Z \cap \left\{p_{\mathsf{Left}(w_j)}, p_{\mathsf{Right}(w_j)}\right\}$ and the output bit 0.

This concludes the analysis of local randomness, addition and multiplication.

Transmit and OLE statements. Assume that the i-th statement is $\mathsf{transmit}(w_{j_1}, w_{j_2}, x^{k_1}, x^{k_2})$ or $\mathsf{func}(\mathsf{OLE}, w_{j_1}, w_{j_2}, (x^{k_1}, x^{k_2}), x^{k_3}, x^{k_4})$. Let us set $\mathcal{P}' := \left\{p_{\mathsf{Left}(w_{j_1})}, p_{\mathsf{Right}(w_{j_1})}, p_{\mathsf{Left}(w_{j_2})}, p_{\mathsf{Right}(w_{j_2})}\right\}$, and denote the fixed inputs of $p_{\mathsf{Left}(w_{j_1})}$ and $p_{\mathsf{Right}(w_{j_1})}$ in the execution of $\mathsf{LROLE}_{\Sigma_1,\Sigma_2}$ by $(\mathbf{a}_0, \mathbf{b}_0)$ and $(\mathbf{a}_1, \mathbf{b}_1)$, respectively, and the fixed inputs of $p_{\mathsf{Left}(w_{j_2})}$ and $p_{\mathsf{Right}(w_{j_2})}$ by $\mathbf{x}_0$ and $\mathbf{x}_1$, respectively.

We denote by v the value of the output variable in $\mathcal{V}^h$. That is, if the i-th statement is $\mathsf{transmit}(w_{j_1}, w_{j_2}, x^{k_1}, x^{k_2})$ then v is the value of x^{k_2} according to $\mathcal{V}^h$, and if the i-th statement is $\mathsf{func}(\mathsf{OLE}, w_{j_1}, w_{j_2}, (x^{k_1}, x^{k_2}), x^{k_3}, x^{k_4})$ then v is the value of x^{k_4} according to $\mathcal{V}^h$. We split into cases.

First case: $w_{j_1}, w_{j_2} \in Z_h$. First, we observe that if $w_{j_1}, w_{j_2} \in Z_h$ then $\mathsf{hybrid}_{i-1} \equiv \mathsf{hybrid}_i$. Indeed, in both cases the execution of statements $i, \ldots, |\pi_h|$ is performed in the same way.

Second Case: ($w_{j_1}, w_{j_2} \notin Z_h$) OR ($w_{j_1} \in Z_h$ and $w_{j_2} \notin Z_h$). Assume that $w_{j_1}, w_{j_2} \notin Z_h$ (resp., $w_{j_1} \in Z_h$ and $w_{j_2} \notin Z_h$). Observe that the compilation of statements $i+1, \ldots, |\pi_h|$ in Sim is independent of the values generated in the compilation of the i-th statement. Therefore, there is a fixed choice of all the variables in statements $i+1, \ldots, |\pi_h|$ such that hybrid'_{i-1} is $1/(\kappa \cdot |\pi_h|)$-far from hybrid'_i. Let us denote the conditional hybrids by hybrid''_{i-1} and hybrid''_i, and observe that the views $(\mathsf{view}_k)_{p_k \notin \mathcal{P}'}$ are fixed, and that for every $p_k \in \mathcal{P}'$ the view view_k is only missing the partial view and output of $\mathsf{LROLE}_{\Sigma_1,\Sigma_2}$ from the compilation of the i-th statement. We also observe that in hybrid''_i the compilation of the i-th statement of π_h corresponds to an execution of $\mathsf{LROLE}_{\Sigma_1,\Sigma_2}$, and that in hybrid''_{i-1} the execution is simulated by sampling the outputs $(\mathsf{out}_0, \mathsf{out}_1) \leftarrow \mathsf{LRShare}_{\Sigma_1}(0)$ and the partial view $\mathsf{Sim}_{\mathsf{LROLE}}(Z \cap \mathcal{P}', 0, \mathsf{aux})$ where $\mathsf{aux} = \emptyset$ (resp., $\mathsf{aux} = ((\mathbf{a}_0, \mathbf{b}_0), (\mathbf{a}_1, \mathbf{b}_1))$). Recall that we've chosen Σ_1 and Σ_2 so that $\mathsf{LROLE}_{\Sigma_1,\Sigma_2}$ would satisfy $(\mu, \frac{1}{\kappa \cdot |\pi_h|})$-composable leakage-resilient passive security. We continue by showing that we can violate the $(\mu, \frac{1}{\kappa \cdot |\pi_h|})$-composable leakage-resilient passive security of $\mathsf{LROLE}_{\Sigma_1,\Sigma_2}$, in contradiction to Lemma 5.2. Indeed, for every $p \in \mathcal{P}'$ define the leakage function ν_p in the following way:

1. The function takes as an input the partial view and output $(\mathsf{pView}_p^{\mathsf{LROLE}}, \mathsf{out}_p^{\mathsf{LROLE}})$.
2. Given the values the we've fixed and $(\mathsf{pView}_p^{\mathsf{LROLE}}, \mathsf{out}_p^{\mathsf{LROLE}})$, the function compute the view view_p.

3. The function outputs $\tau_p(\mathsf{view}_p)$.

It is not hard to verify that the composable leakage-resilient passive security of $\mathsf{LROLE}_{\Sigma_1,\Sigma_2}$ is violated with respect to the inputs $(\mathbf{a}_0, \mathbf{b}_0)$, $(\mathbf{a}_1, \mathbf{b}_1)$, $\mathbf{x}_0$ and $\mathbf{x}_1$, the leakage functions $\{\nu_p\}_{p\in\mathcal{P}'}$, the set $Z \cap \mathcal{P}'$ and the output bit 0.

Third case: $w_{j_1} \notin Z_h$ and $w_{j_2} \in Z_h$. It remains to consider the case where $w_{j_1} \notin Z_h$ and $w_{j_2} \in Z_h$. Let us denote the output of the receivers in the $\mathsf{LROLE}_{\Sigma_1,\Sigma_2}$ instance by $(\mathsf{out}_0, \mathsf{out}_1)$ and the partial views of the parties by $(\mathsf{pView}_p^{\mathsf{LROLE}})_{p\in\mathcal{P}'}$. First, let us observe that in both hybrids $(\mathsf{out}_0, \mathsf{out}_1)$ have the same distributions, and that it is independent of the partial views $(\mathsf{pView}_p^{\mathsf{LROLE}})_{p\in\mathcal{P}'}$. Indeed, if the i-th statement is $\mathsf{transmit}(w_{j_1}, w_{j_2}, x^{k_1}, x^{k_2})$ then the value $\mathsf{val}(x^{k_2})$ according to $\mathcal{V}^h$ is the same in both hybrids, in hybrid'_{i-1} by definition of Sim, it holds that $(\mathsf{out}_0, \mathsf{out}_1) \leftarrow \mathsf{LRShare}_{\Sigma_1}(\mathsf{val}(x^{k_2}))$ and they are independent of $(\mathsf{pView}_p^{\mathsf{LROLE}})_{p\in\mathcal{P}'}$, and in hybrid'_i, by the real-world output independence guaranteed by the leakage-resilient composable security of LROLE, it holds that $(\mathsf{out}_0, \mathsf{out}_1) \leftarrow \mathsf{LRShare}_{\Sigma_1}(\mathsf{val}(x^{k_2}))$ and independent of $(\mathsf{pView}_p^{\mathsf{LROLE}})_{p\in\mathcal{P}'}$. A similar argument shows that the same is true for an OLE-statement. Let us denote the fixed outputs by $(\mathsf{out}_0^*, \mathsf{out}_1^*)$.

We also observe that the compilation of statements $i+1, \ldots, |\pi_h|$ in Sim is independent of the values generated in the compilation of the i-th statement, except for the outputs $(\mathsf{out}_0, \mathsf{out}_1)$. Therefore, there is a fixed choice of the outputs $(\mathsf{out}_0, \mathsf{out}_1)$ (that we denoted by $(\mathsf{out}_0^*, \mathsf{out}_1^*)$) and all the variables in statements $i+1, \ldots, |\pi_h|$ such that hybrid'_{i-1} is $1/(\kappa \cdot |\pi_h|)$-far from hybrid'_i. Let us denote the conditional hybrids by hybrid''_{i-1} and hybrid''_i, and observe that the views $(\mathsf{view}_k)_{p_k\notin\mathcal{P}'}$ are fixed, and that for every $p_k \in \mathcal{P}'$ the view view_k is only missing the partial view of $\mathsf{LROLE}_{\Sigma_1,\Sigma_2}$ from the compilation of the i-th statement. We also observe that in hybrid''_i the compilation of the i-th statement of π_h corresponds to an execution of $\mathsf{LROLE}_{\Sigma_1,\Sigma_2}$, and that in hybrid''_{i-1} the execution corresponds to $\mathsf{Sim}_{\mathsf{LROLE}}(Z \cap \mathcal{P}', v, \mathsf{aux})$ where $\mathsf{aux} = (\mathbf{x}_0, \mathbf{x}_1)$. Recall that we've chosen Σ_1 and Σ_2 so that $\mathsf{LROLE}_{\Sigma_1,\Sigma_2}$ would satisfy $(\mu, \frac{1}{\kappa\cdot|\pi_h|})$-composable leakage-resilient passive security. We continue by showing that we can violate the $(\mu, \frac{1}{\kappa\cdot|\pi_h|})$-composable leakage-resilient passive security of $\mathsf{LROLE}_{\Sigma_1,\Sigma_2}$, in contradiction to Lemma 5.2. In fact, our leakage function only requires the partial view of the party, without the output. Indeed, for every $p \in \mathcal{P}'$ define the leakage function ν_p in the following way:

1. The function takes as an input the partial view and output $(\mathsf{pView}_p^{\mathsf{LROLE}})$.
2. Given the values the we've fixed and $(\mathsf{pView}_p^{\mathsf{LROLE}})$, the function computes the view view_p.
3. The function outputs $\tau_p(\mathsf{view}_p)$.

It is not hard to verify that the composable leakage-resilient passive security of $\mathsf{LROLE}_{\Sigma_1,\Sigma_2}$ is violated with respect to the inputs $(\mathbf{a}_0, \mathbf{b}_0)$, $(\mathbf{a}_1, \mathbf{b}_1)$, $\mathbf{x}_0$ and $\mathbf{x}_1$, the leakage functions $\{\nu_p\}_{p\in\mathcal{P}'}$ and the set $Z \cap \mathcal{P}'$.

This concludes the proof of the lemma. □

B.4 Proof of Lemma 5.8

Fix any set $S \subseteq \{p_1, \ldots, p_n\}$ that satisfies $\chi(S) = 1$, any encoding $(\mathsf{input}_1, \ldots, \mathsf{input}_n)$, and any adversarial behavior $\tilde{\pi}$ with respect to the set $Z := \{p_1, \ldots, p_n\} \setminus S$.

We observe that any choice of $(\mathsf{input}_1, \ldots, \mathsf{input}_n)$ with the appropriate length is a valid encoding, i.e., there exists some input $\mathbf{x} \in \{0,1\}^{\ell_{\mathsf{input}}}$ such that $(\mathsf{input}_1, \ldots, \mathsf{input}_n)$ is in the support of $\mathsf{Enc}(\mathbf{x})$. This follows since the encoding is generated by computing n-out-of-n secret sharing of $\mathbf{x}$, denoted $\mathbf{x}^1, \ldots, \mathbf{x}^m$, and then Σ_1-sharing every bit of $(\mathbf{x}^1, \ldots, \mathbf{x}^m)$ between some pair of parties. As we've mentioned in Section 4.2, every pair of shares of appropriate length $(\mathbf{s}_0, \mathbf{s}_1)$ is a valid Σ_1-sharing for some bit $s \in \{0,1\}$, and therefore any choice of $(\mathsf{input}_1, \ldots, \mathsf{input}_n)$ with the appropriate length is a valid encoding. Therefore, we may assume without loss of generality that $(\mathsf{input}_1, \ldots, \mathsf{input}_n)$ form a valid encoding of some input $\mathbf{x} \in \{0,1\}^{\ell_{\mathsf{input}}}$, as we can always let the parties pad the input with zeros if it is too short, or ignore the suffix if it is too long.

Consider any execution of $\tilde{\pi}$ with the inputs $(\mathsf{input}_1, \ldots, \mathsf{input}_n)$ in which the abort flag was not raised, and let $(\mathsf{view}_i)_{p_i \in S}$ be the corresponding views of the honest parties. Let S_h and (view'_i) be defined as in Ext, and let $Z_h := \{w_1, \ldots, w_m\} \setminus S_h$ be the set corresponding to the corrupt parties in π_h. Since $\chi(S) = 1$ and $\chi := (h, \mathsf{AND}_2^m, T)$, then $1 = \chi(S) = h(S_h)$. We define the adversarial behavior $\tilde{\pi}_h$ with respect to Z_h in the following way:

- In every statement $\mathsf{transmit}(w_{j_1}, w_{j_2}, x^{k_1}, x^{k_2})$ where $w_{j_1} \in Z_h$ and $w_{j_2} \in S_h$, mark the statement with the value of the variable x^{k_2} according to view'_{j_2}.
- In every statement $\mathsf{func}(\mathsf{OLE}, w_{j_1}, w_{j_2}, (x^{k_1}, x^{k_2}), x^{k_3}, x^{k_4})$, where $w_{j_1} \in Z_h$ and $w_{j_2} \in S_h$, mark the statement with the value of the variable x^{k_4} according to view'_{j_2}.

It is not hard to see that when we execute $\tilde{\pi}_h$, and the parties in S_h pick the same randomness as in $(\mathsf{view}'_i)_{w_i \in S_h}$, then we obtain the views $(\mathsf{view}'_i)_{w_i \in S_h}$. In addition, since no $p_i \in S$ raises the abort flag according to $(\mathsf{view}_i)_{p_i \in S}$, then no party in $w_i \in S_h$ raises the abort flag according to $(\mathsf{view}'_i)_{w_i \in S_h}$. Therefore, by the h-perfect active correctness with abort of π_h, it holds that the outputs of the parties in S_h is consistent with the inputs $(\mathbf{x}^1, \ldots, \mathbf{x}^m) := \mathsf{Ext}_h(S_h, (\mathsf{view}'_i)_{w_i \in S_h})$, i.e., the output is $f(\mathbf{x}^1 \oplus \ldots \oplus \mathbf{x}^m)$. Since the output of the w_i's in S_h is the same as the output of the p_i's in S, it holds that the output of the p_i's is indeed $f(\mathbf{x})$ where $\mathbf{x} = \mathbf{x}^1 \oplus \ldots \oplus \mathbf{x}^m$. This concludes the proof of the lemma.

B.5 Proof of Lemma 6.4

One can verify that perfect correctness follows from the perfect correctness of the original protocol (π', Enc') and the perfect correctness of the PSM (Lemma 6.2). We continue by analyzing statistical privacy. For this, fix any input $\mathbf{x} \in \{0,1\}^{\ell_{\mathsf{input}}}$, any functions $\tau_1, \ldots, \tau_n$ that output μ bits, and any set $Z \subseteq \mathcal{P}$ such that $\chi(Z) = 0$. Let us denote by M the number of func-statements. Since $M \leq |\pi'|$ and $2\ell \cdot \ell_{\mathsf{psm}} \leq s$, it is enough to prove that Π computes f with (χ, μ, ϵ)-leakage-resilient passive security for $\epsilon = \epsilon' + M \cdot 2\ell \cdot \ell_{\mathsf{psm}} \cdot \epsilon_\Sigma$.

Assume towards contradiction that

$$(\mathsf{view}_i)_{p_i \in Z}, (\tau_i(\mathsf{view}_i))_{p_i \notin Z} \quad \text{is } \epsilon\text{-far from} \quad (\mathsf{view}'_i)_{p_i \in Z}, (\tau_i(\mathsf{view}'_i))_{p_i \notin Z}, \tag{9}$$

where $(\mathsf{view}_1, \ldots, \mathsf{view}_n)$ were sampled in an honest execution of Π on input $\mathbf{x}$, and $(\mathsf{view}'_1, \ldots, \mathsf{view}'_n)$ were sampled from $\mathsf{Sim}(Z, f(\mathbf{x}))$. We prove that there exist leakage functions $\nu_1, \ldots, \nu_n$ such that the (χ, μ, ϵ')-leakage-resilient passive security of Π' is violated with respect to $\mathbf{x}$, Z and $(\nu_1, \ldots, \nu_n)$. At a high level, we show that for every $\mathcal{F}_i$-statement in π', we can use the leakage function to emulate the views of the parties in the computation of $\mathcal{F}_i$ in π.

The leakage functions ν_i. Let $\mathbf{r}$ be a random vector, that for every $\mathcal{F}$-statement $\mathsf{func}(\mathcal{F}, p, q, X, Y, Z)$ in π' contains the tuple

$$(\mathbf{r}_{\mathsf{psm}}, (\sigma'_j)_{j\in[2\ell]}, (A_{j,0}, A'_{j,1})_{j\in[2\ell]}, (B_{j,0}, B_{j,1})_{j\in[2\ell]}),$$

where (1) $\mathbf{r}_{\mathsf{psm}}$ is randomness for the simulator $\mathsf{Sim}_{\mathsf{psm}}$ of the PSM scheme of $\mathcal{F}$, (2) $A_{j,0}$ is a vector of ℓ_{psm} shares, each sampled from LRRand_Σ, (3) $A'_{j,1}$ is a vector of ℓ_{psm} partial shares, each sampled from $\mathsf{LRPartial}_\Sigma$, (4) $(B_{j,0}, B_{j,1}) \leftarrow \mathsf{LRvShare}_\Sigma(\vec{0})$, where $\vec{0}$ is a vector of ℓ_{psm} zeros.

For every $\mathbf{r}$ and every $i \in [n]$ define the leakage function $\nu_i^{\mathbf{r}}$ in the following way:

1. The leakage function takes as an input a view view_i corresponding to the view of the i-th party in an execution of Π'.
2. For every $\mathcal{F}$-statement $\mathsf{func}(\mathcal{F}, p_i, q, X, Y, Z)$ in π', the function generates the view of p_i that corresponds to the computation of the $\mathcal{F}$-statement in π in the following way:
   - Denote the input of p_i to $\mathcal{F}$ according to view_i by $\mathbf{w}$.
   - Let $(\mathbf{r}_{\mathsf{psm}}, (\sigma'_j)_{j\in[2\ell]}, (A_{j,0}, A'_{j,1})_{j\in[2\ell]}, (B_{j,0}, B_{j,1})_{j\in[2\ell]})$ be the tuple from $\mathbf{r}$ that corresponds to the current statement.
   - For every $j \in [\ell]$, set $M_{j,w_j,0} := A_{j,0}$, $M_{j,1\oplus w_j,0} := B_{j,0}$ and $\sigma_j := \sigma'_j \oplus w_j$. The corresponding outgoing message is σ'_j.
   - For every $j \in \{\ell+1, \ldots, 2\ell\}$, the party p_i should hold a permuted pair $(M'_{j,0,0}, M'_{j,1,0})$ (where in an honest execution $M'_{j,0,0} = M_{j,0\oplus\sigma_j,0}$ and $M'_{j,1,0} = M_{j,1\oplus\sigma_j,0}$). We set $M'_{j,\sigma'_j,0} := A_{j,0}$, $M'_{j,1\oplus\sigma'_j,0} := B_{j,0}$, and let σ'_j be the corresponding incoming message from q.
3. For every $\mathcal{F}$-statement $\mathsf{func}(\mathcal{F}, q, p_i, X, Y, Z)$ in π', the function generates the view of p_i that corresponds to the computation of the $\mathcal{F}$-statement in π in the following way:
   - Denote the input of p_i to $\mathcal{F}$ according to view_i by $\mathbf{y}$, and the output by $\mathbf{z}$.
   - Let $(\mathbf{r}_{\mathsf{psm}}, (\sigma'_j)_{j\in[2\ell]}, (A_{j,0}, A'_{j,1})_{j\in[2\ell]}, (B_{j,0}, B_{j,1})_{j\in[2\ell]})$ be the tuple from $\mathbf{r}$ that corresponds to the current statement.
   - Compute $(\mathbf{m}_1, \ldots, \mathbf{m}_{2\ell}) := \mathsf{Sim}_{\mathsf{psm}}(\mathbf{z}; \mathbf{r}_{\mathsf{psm}})$.
   - For every $j \in [\ell]$, the party p_i should hold a permuted pair $(M'_{j,0,1}, M'_{j,1,1})$ (where in an honest execution $M'_{j,0,1} = M_{j,0\oplus\sigma_j,1}$ and $M'_{j,1,1} = M_{j,1\oplus\sigma_j,1}$). We set $M'_{j,\sigma'_j,1} := (A'_{j,1}, \mathsf{LRvUpdate}_\Sigma(\mathbf{m}_j, A_{j,0}, A'_{j,1}))$, $M'_{j,1\oplus\sigma'_j,1} := B_{j,1}$ and set the incoming messages to be σ'_j.
   - For every $j \in \{\ell+1, \ldots, 2\ell\}$, set $M_{j,y_{j-\ell},1} := (A'_{j,1}, \mathsf{LRvUpdate}_\Sigma(\mathbf{m}_j, A_{j,0}, A'_{j,1}))$, $M_{j,1\oplus y_{j-\ell},1} := B_{j,1}$ and $\sigma_j := \sigma'_j \oplus y_{j-\ell}$. The corresponding outgoing message is σ'_j.
4. The function updates the view view_i accordingly, and outputs $\tau_i(\mathsf{view}_i)$.

The privacy of Π'. The privacy of Π' implies that

$$\mathbf{r}, (\widetilde{\mathsf{view}}_i)_{p_i \in Z}, (\nu_i^{\mathbf{r}}(\widetilde{\mathsf{view}}_i))_{p_i \notin Z} \equiv_{\epsilon'} \mathbf{r}, (\widetilde{\mathsf{view}}'_i)_{p_i \in Z}, (\nu_i^{\mathbf{r}}(\widetilde{\mathsf{view}}'_i))_{p_i \notin Z}, \tag{10}$$

for every $\mathbf{r}$, where $(\widetilde{\mathsf{view}}_1, \ldots, \widetilde{\mathsf{view}}_n)$ were sampled in an honest execution of Π' on input $\mathbf{x}$, and $(\widetilde{\mathsf{view}}'_1, \ldots, \widetilde{\mathsf{view}}'_n)$ were sampled from $\mathsf{Sim}'(Z, f(\mathbf{x}))$. In particular, Equation 10 holds even when $\mathbf{r}$ is chosen at random.

The procedure $\mathcal{M}$. Consider the following deterministic procedure $\mathcal{M}$.

Procedure $\mathcal{M}$

- **Inputs:** $\mathcal{M}$ takes as an input a tuple $(\mathbf{r}, (\mathsf{view}_i)_{p_i \in Z}, (\mathsf{leak}_i)_{p_i \notin Z})$.
- **Computation:** For every $\mathcal{F}$-statement $\mathsf{func}(\mathcal{F}, p_i, p_{i'}, X, Y, Z)$ in π', the procedure $\mathcal{M}$ updates the views of p_i and $p_{i'}$ by adding an emulation to the computation of the $\mathcal{F}$-statement in π in the following way:

  Case 1: $p_i, p_{i'} \in Z$. First, we consider the case where $p_i, p_{i'} \in Z$. Let us denote the inputs of p_i and $p_{i'}$ to $\mathcal{F}$ according to view_i and $\mathsf{view}_{i'}$ by $\mathbf{w}$ and $\mathbf{y}$, respectively. In addition, we denote the output of $p_{i'}$ according to $\mathsf{view}_{i'}$ by $\mathbf{z}$. In this case, the procedure samples the correlated randomness $(M_{j,b,0})_{j\in[\ell],b\in\{0,1\}}, (M_{j,b\oplus\sigma_j,0})_{j\in[\ell],b\in\{0,1\}}, (\sigma_j)_{j\in[\ell]}$ for p_i and $(M_{j,b\oplus\sigma_j,1})_{j\in[\ell],b\in\{0,1\}}, (M_{j,b,1})_{j\in\{\ell+1,\ldots,2\ell\},b\in\{0,1\}}, (\sigma_j)_{j\in\{\ell+1,\ldots,2\ell\}}$ for $p_{i'}$, as described in Figure 15. Then, the procedure updates the views of p_i and $p_{i'}$ by executing the sub-protocol from Figure 16 with the correlated randomness it sampled and inputs $\mathbf{w}$ and $\mathbf{y}$. This concludes the update procedure for the case where $p_i, p_{i'} \in Z$.

  Case 2: $p_i \in Z$ and $p_{i'} \notin Z$. Assume that $p_i \in Z$ and $p_{i'} \notin Z$. In this case, the procedure updates only the view of p_i in the following way. Denote the input of p_i to $\mathcal{F}$ according to view_i by $\mathbf{w}$. Let $(\mathbf{r}_{\mathsf{psm}}, (\sigma'_j)_{j\in[2\ell]}, (A_{j,0}, A'_{j,1})_{j\in[2\ell]}, (B_{j,0}, B_{j,1})_{j\in[2\ell]})$ be the tuple from $\mathbf{r}$ that corresponds to the current statement. For every $j \in [\ell]$, set $M_{j,w_j,0} := A_{j,0}$, $M_{j,1\oplus w_j,0} := B_{j,0}$ and $\sigma_j := \sigma'_j \oplus w_j$. For every $j \in \{\ell+1, \ldots, 2\ell\}$ define the permuted pair $(M'_{j,0,0}, M'_{j,1,0})$ by $M'_{j,\sigma'_j,0} := A_{j,0}$, $M'_{j,1\oplus\sigma'_j,0} := B_{j,0}$, and let σ'_j be the corresponding incoming message from $p_{i'}$.

  Case 3: $p_i \notin Z$ and $p_{i'} \in Z$. Assume that $p_i \notin Z$ and $p_{i'} \in Z$. In this case, the procedure updates only the view of $p_{i'}$ in the following way. Denote the input of $p_{i'}$ to $\mathcal{F}$ according to $\mathsf{view}_{i'}$ by $\mathbf{y}$, and the output by $\mathbf{z}$. Let $(\mathbf{r}_{\mathsf{psm}}, (\sigma'_j)_{j\in[2\ell]}, (A_{j,0}, A'_{j,1})_{j\in[2\ell]}, (B_{j,0}, B_{j,1})_{j\in[2\ell]})$ be the tuple from $\mathbf{r}$ that corresponds to the current statement. Compute $(\mathbf{m}_1, \ldots, \mathbf{m}_{2\ell}) := \mathsf{Sim}_{\mathsf{psm}}(\mathbf{z}; \mathbf{r}_{\mathsf{psm}})$. For every $j \in [\ell]$, define the permuted pair $(M'_{j,0,1}, M'_{j,1,1})$ by $M_{j,\sigma'_j,1} := (A'_{j,1}, \mathsf{LRvUpdate}_\Sigma(\mathbf{m}_j, A_{j,0}, A'_{j,1}))$, $M_{j,1\oplus\sigma'_j,1} := B_{j,1}$ and set the incoming messages to be σ'_j. For every $j \in \{\ell+1, \ldots, 2\ell\}$, set $M_{j,y_{j-\ell},1} := (A'_{j,1}, \mathsf{LRvUpdate}_\Sigma(\mathbf{m}_j, A_{j,0}, A'_{j,1}))$, and $M_{j,1\oplus y_{j-\ell},1} := B_{j,1}$.
- **Outputs:** After updating the views $(\mathsf{view}_i)_{p_i\in Z}$, the procedure outputs $(\mathsf{view}_i)_{p_i\in Z}, (\mathsf{leak}_i)_{p_i\notin Z}$.

Figure 22: Procedure $\mathcal{M}$

First, observe that if we apply $\mathcal{M}$ on the RHS of Equation 10 when $\mathbf{r}$ is uniformly distributed, then we obtain the same distribution as the RHS of Equation 9. Therefore, to get a contradiction it only remains to prove that if we apply $\mathcal{M}$ on the LHS of Equation 10 then we obtain a distribution that is statistically-close to the LHS of Equation 9. We prove this in the following claim.

Claim B.13. *It holds that*

$$(\mathsf{view}_i)_{p_i\in Z}, (\tau_i(\mathsf{view}_i))_{p_i\notin Z} \equiv_{M\cdot 2\ell\cdot \ell_{\mathsf{psm}}\cdot\epsilon_\Sigma} \mathcal{M}(\mathbf{r}, (\widetilde{\mathsf{view}}_i)_{p_i\in Z}, (\nu_i^{\mathbf{r}}(\widetilde{\mathsf{view}}_i))_{p_i\notin Z}),$$

where on the LHS $(\mathsf{view}_1,\ldots,\mathsf{view}_n)$ *are sampled in an honest execution of* Π *on input* $\mathbf{x}$*, and on the RHS* $\mathbf{r}$ *is chosen at random and* $(\widetilde{\mathsf{view}}_1,\ldots,\widetilde{\mathsf{view}}_n)$ *are sampled in an honest execution of* Π' *on input* $\mathbf{x}$*.*

Proof of Claim B.13. First, let us consider the following hybrid.

Hybrid. In hybrid we generate the views $\mathsf{view}_1,\ldots,\mathsf{view}_n$ by executing the protocol Π on input $\mathbf{x}$, except that for any $\mathcal{F}_i$-statement $\mathsf{func}(\mathcal{F}_i,p,q,X,Y,Z)$ with $p\notin Z$ or $q\notin Z$, the correlated randomness is sampled adaptively in the following way: if p holds input $\mathbf{w}$ and q holds input and $\mathbf{y}$, we sample the correlated randomness by first computing $\mathbf{z}=\mathcal{F}_i(\mathbf{w},\mathbf{y})$ and sampling $(\mathbf{m}_1,\ldots,\mathbf{m}_{2\ell})\leftarrow \mathsf{Sim}_{\mathsf{psm}}(\mathbf{z})$. Then, for every $j\in[2\ell]$ we sample $(A_{j,0},A_{j,1})\leftarrow \mathsf{LRvShare}(\mathbf{m}_j)$ and $(B_{j,0},B_{j,1})\leftarrow \mathsf{LRvShare}(\vec{0})$, where $\vec{0}$ is a vector of ℓ_{psm} zeros. We also samples random bits $\sigma_1,\ldots,\sigma_{2\ell}$. For every $j\in[\ell]$ we set $M_{j,w_j,0}:=A_{j,0}$, $M_{j,w_j,1}:=A_{j,1}$, $M_{j,1\oplus w_j,0}:=B_{j,0}$, and $M_{j,1\oplus w_j,1}:=B_{j,1}$. For every $j\in\{\ell+1,\ldots,2\ell\}$ we set $M_{j,y_{j-\ell},0}:=A_{j,0}$, $M_{j,y_{j-\ell},1}:=A_{j,1}$, $M_{j,1\oplus y_{j-\ell},0}:=B_{j,0}$, $M_{j,1\oplus y_{j-\ell},1}:=B_{j,1}$. We set the correlated randomness of p and q to be $(M_{j,b,0})_{j\in[\ell],b\in\{0,1\}}$, $(M_{j,b\oplus\sigma_j,0})_{j\in\{\ell+1,\ldots,2\ell\},b\in\{0,1\}}$, $(\sigma_j)_{j\in[\ell]}$ for p and $(M_{j,b\oplus\sigma_j,1})_{j\in[\ell],b\in\{0,1\}}$, $(M_{j,b,1})_{j\in\{\ell+1,\ldots,2\ell\},b\in\{0,1\}}$, $(\sigma_j)_{j\in\{\ell+1,\ldots,2\ell\}}$ for q. We then execute the sub-protocol with inputs $\mathbf{w}$ and $\mathbf{y}$ and with the above correlated randomness. We emphasize that if $p,q\in Z$ then the correlated randomness is sampled according to Rand.

Analysis. Let us denote the views generated in hybrid by $\mathsf{view}'_1,\ldots,\mathsf{view}'_n$. It is not hard to verify that

$$(\mathsf{view}'_i)_{p_i\in Z}, (\tau_i(\mathsf{view}'_i))_{p_i\notin Z} \equiv \mathcal{M}(\mathbf{r}, (\widetilde{\mathsf{view}}_i)_{p_i\in Z}, (\nu_i^{\mathbf{r}}(\widetilde{\mathsf{view}}_i))_{p_i\notin Z}).$$

We continue by comparing hybrid to an execution of Π on $\mathbf{x}$. First, the perfect privacy of the PSM implies that in execution the sub-protocol for the computation of $\mathcal{F}_i$ with the parties p and q on inputs $\mathbf{w}$ and $\mathbf{y}$, the random variables $(M_{j,w_j,0},M_{j,w_j,1})_{j\in[\ell]}$ and $(M_{j,y_{j-\ell},0},M_{j,y_{j-\ell},1})_{j\in\{\ell+1,\ldots,2\ell\}}$ have the same distribution in $(\mathsf{view}_i)_{i\in[n]}$ and $(\mathsf{view}'_i)_{i\in[n]}$. Therefore, the only difference is that in a real-world execution the random variables $(M_{j,w_j\oplus 1,0},M_{j,w_j\oplus 1,1})_{j\in[\ell]}$ and $(M_{j,y_{j-\ell}\oplus 1,0},M_{j,y_{j-\ell}\oplus 1,1})_{j\in\{\ell+1,\ldots,2\ell\}}$ correspond to a sharing of a real PSM messages, while in hybrid they correspond to the all-zero messages. Therefore, a standard hybrid argument, similar to that of Claim B.1, shows that since Σ is (μ,ϵ_Σ)-local leakage resilient then

$$(\mathsf{view}_i)_{p_i\in Z}, (\tau_i(\mathsf{view}_i))_{p_i\notin Z} \equiv_{M\cdot 2\ell\cdot \ell_{\mathsf{psm}}\cdot\epsilon_\Sigma} (\mathsf{view}'_i)_{p_i\in Z}, (\tau_i(\mathsf{view}'_i))_{p_i\notin Z},$$

where we used the fact that the number of $\mathcal{F}$-statements in π' is M and that $(M_{j,w_j\oplus 1,0},M_{j,w_j\oplus 1,1})_{j\in[\ell]}$ and $(M_{j,y_{j-\ell}\oplus 1,0},M_{j,y_{j-\ell}\oplus 1,1})_{j\in\{\ell+1,\ldots,2\ell\}}$ correspond to Σ-shares of $2\ell\cdot\ell_{\mathsf{psm}}$ bits. This concludes the proof of the claim. □

This concludes the proof of the lemma. □

C Missing Proofs: NIZK

C.1 Proof of Theorem 8.5

First, We need the following version of Yao's XOR lemma, that can be obtained by slightly modifying the proof of [GNW11, Lemma 1 and Lemma 2].

Theorem C.1 (Yao's XOR lemma). *Let $\ell : \mathbb{N} \to \mathbb{N}$ be a function, let $\mathbf{X} = \{X_\kappa\}_{\kappa\in\mathbb{N}}$ be a probability ensemble such that, for every κ, the random variable X_κ is distributed over $\{0,1\}^{\ell(\kappa)}$, and let $\{P_\kappa\}_{\kappa\in\mathbb{N}}$ be a family of randomized algorithms where P_κ takes length-$\ell(\kappa)$ bit strings as an input and outputs a bit. For every function $n : \mathbb{N} \to \mathbb{N}$ and every $\kappa \in \mathbb{N}$ define the predicate*

$$P_\kappa^{(n(\kappa))}(x_1, \ldots, x_{n(\kappa)}) = \bigoplus_{i=1}^{n(\kappa)} P_\kappa(x_i),$$

where $x_1, \ldots, x_{n(\kappa)} \in \{0,1\}^{\ell(\kappa)}$ and let $\mathbf{X}_\kappa^{(n)} = \left\{X_\kappa^{(n)}\right\}_{\kappa\in\mathbb{N}}$ be a probability ensemble such that $X_\kappa^{(n)}$ consists of $n(\kappa)$ independent copies of X_κ.

Let $\sigma : \mathbb{N} \to \mathbb{N}$ be a size function, and $\delta : \mathbb{N} \to [0,1]$ be a function that is bounded-away-from-1 *(i.e., $\delta(\kappa) < 1 - \frac{1}{p(\kappa)}$ for some polynomial p and all sufficiently large κ's). Suppose that for every size-σ family of circuits $\{C_\kappa\}_{\kappa\in\mathbb{N}}$ and all κ it holds that*

$$2 \cdot \Pr[C_\kappa(X_\kappa) = P_\kappa(X_\kappa)] - 1 \leq \delta(\kappa).$$

Then for every function $n : \mathbb{N} \to \mathbb{N}$, every function $\epsilon : \mathbb{N} \to [0,1]$, every size-$\sigma'$ family of circuits $\{C'_\kappa\}_{\kappa\in\mathbb{N}}$ and all κ it holds that

$$2 \cdot \Pr\left[C'_\kappa\left(X_\kappa^{(n)}\right) = P_\kappa^{(n)}\left(X_\kappa^{(n)}\right)\right] - 1 \leq \delta^{(n)}(\kappa),$$

where $\delta^{(n)}(\kappa) = \delta(\kappa)^{n(\kappa)} + \epsilon(\kappa)$ and

$$\sigma'(\kappa) = \frac{\sigma(\kappa)}{\mathrm{poly}(\ell(\kappa) \cdot n(\kappa)/\epsilon(\kappa))} - \mathrm{poly}(n(\kappa) \cdot \ell(\kappa)).$$

We continue with the proof of Theorem 8.5.

Proof of Theorem 8.5. Fix functions n and ϵ, and assume towards contradiction that there is a size-s' adversary $\mathcal{D} = \{\mathcal{D}_\kappa\}_\kappa$ such that $\mathsf{Guess}^{\mathcal{D}_\kappa}(A_\kappa, B_\kappa, n(\kappa)) > \delta'(\kappa)$ for some κ. We may assume without loss of generality that $\mathcal{D}$ is deterministic, and since the queries to the oracles are not adaptive, let us denote the queries of $\mathcal{D}_\kappa$ to $F_1, \ldots, F_{n(\kappa)}$ by $q_\kappa^1, \ldots, q_\kappa^{n(\kappa)}$. We show how to use $\mathcal{D}$ to contradict Yao's XOR lemma.

From guessing games to Yao's XOR lemma. Let Exp_κ be the experiment where (1) we pick a random bit $b \leftarrow \{0,1\}$ and index $i \leftarrow [n(\kappa)]$, (2) if $b = 0$ then we sample $\mathbf{y} \leftarrow A_\kappa(q_\kappa^i)$ and if $b = 1$ then we sample $\mathbf{y} \leftarrow B_\kappa(q_\kappa^i)$, and (3) we output $(b, i, \mathbf{y})$. Define X_κ to be the following distribution over $\{0,1\}^{\ell_{\mathsf{out}}(\kappa)+\lceil\log(n(\kappa))+1\rceil}$: sample $(b, i, \mathbf{y})$ like in Exp_κ and output $(i, \mathbf{y})$. Let P_κ be the randomized algorithm that given $(i, \mathbf{y})$ samples $(b', i', \mathbf{y}')$ like in Exp_κ conditioned on $(i', \mathbf{y}') = (i, \mathbf{y})$, and outputs b'. Observe that $\mathsf{Exp}_\kappa \equiv (P_\kappa(X_\kappa), X_\kappa)$, and since $\mathsf{Guess}^{s(\kappa)}(A_\kappa, B_\kappa, 1) \leq \delta(\kappa)$ for all $\kappa \in \mathbb{N}$, then for every σ-size adversary $\{C_\kappa\}_{\kappa\in\mathbb{N}}$ and all κ it holds that $2 \cdot \Pr[C_\kappa(X_\kappa) = P_\kappa(X_\kappa)] - 1 \leq \delta(\kappa)$, where $\sigma(\kappa) := s(\kappa) - \mathrm{poly}(\ell_{\mathsf{input}}(\kappa) \cdot \ell_{\mathsf{out}}(\kappa) \cdot n(\kappa))$.

Contradicting the XOR lemma. We define an adversary $\mathcal{E} = \{\mathcal{E}_\kappa\}_{\kappa\in\mathbb{N}}$ where $\mathcal{E}_\kappa$ is defined as follows: (1) it takes as an input $n(\kappa)$ pairs $(i_j, \mathbf{y}_j)_{j\in[n(\kappa)]}$, (2) it verifies that $(i_1, \ldots, i_{n(\kappa)})$ is a permutation of $(1, \ldots, n(\kappa))$, if the verification fails it outputs a random bit and terminates, and otherwise

let us denote the permutation by π (i.e., $i_{\pi(j)} = j$), (3) it executes $\mathcal{D}_\kappa$ as if the answer from F_j is $\mathbf{y}_{\pi(j)}$ and outputs the same bit as $\mathcal{D}_\kappa$.

Observe that $|\mathcal{E}_\kappa| = |\mathcal{D}_\kappa| + \mathrm{poly}(\ell_{\mathsf{out}}(\kappa) \cdot n(\kappa)) \leq \sigma'(\kappa)$, where σ' is as defined in Theorem C.1 with respect to $\sigma(\kappa)$, $\ell(\kappa) := \ell_{\mathsf{out}}(\kappa) + \lceil \log(n(\kappa)) + 1 \rceil$, $n(\kappa)$ and $\epsilon(\kappa)$. Further observe that the probability that $i_1, \ldots, i_{n(\kappa)}$ are distinct is $n(\kappa)!/n(\kappa)^{n(\kappa)} \geq 1/e^{n(\kappa)-1}$ where we used the fact that $n^n/e^{n-1} \leq n!$. Therefore,

$$\begin{aligned}
\Pr[\mathcal{E}_\kappa(X_\kappa^{(n)}) = P_\kappa^{(n)}(X_\kappa^{(n)})] &= \Pr[\mathcal{E}_\kappa((i_j, \mathbf{y}_j)_{j\in[n(\kappa)]}) = P_\kappa^{(n)}((i_j, \mathbf{y}_j)_{j\in[n(\kappa)]})] \\
&= \Pr[\mathcal{E}_\kappa((i_j, \mathbf{y}_j)_{j\in[n(\kappa)]}) = P_\kappa^{(n)}((i_j, \mathbf{y}_j)_{j\in[n(\kappa)]}) \mid i_1, \ldots, i_{n(\kappa)} \text{ are distinct}] \cdot \frac{n(\kappa)!}{n(\kappa)^{n(\kappa)}} \\
&\quad + \frac{1}{2} \cdot \left(1 - \frac{n(\kappa)!}{n(\kappa)^{n(\kappa)}}\right) \\
&= \Pr\left[\mathcal{D}_\kappa^{F_1,\ldots,F_{n(\kappa)}} = \bigoplus_{i\in[n(\kappa)]} s_i\right] \cdot \frac{n(\kappa)!}{n(\kappa)^{n(\kappa)}} + \frac{1}{2} \cdot \left(1 - \frac{n(\kappa)!}{n(\kappa)^{n(\kappa)}}\right) \\
&> \frac{\delta'(\kappa)+1}{2} \cdot \frac{n(\kappa)!}{n(\kappa)^{n(\kappa)}} + \frac{1}{2} \cdot \left(1 - \frac{n(\kappa)!}{n(\kappa)^{n(\kappa)}}\right) \\
&= \frac{1}{2} + \frac{\delta'(\kappa)}{2} \cdot \frac{n(\kappa)!}{n(\kappa)^{n(\kappa)}} \\
&\geq \frac{1}{2} + \frac{\delta'(\kappa)}{2e^{n(\kappa)-1}}.
\end{aligned}$$

But this means that

$$2 \cdot \Pr[\mathcal{E}_\kappa(X_\kappa^{(n)}) = P_\kappa(X_\kappa^{(n)})] - 1 > \delta'(\kappa)/e^{n(\kappa)-1} = \delta(\kappa)^{n(\kappa)} + \epsilon(\kappa),$$

in contradiction to Theorem C.1. This concludes the proof of the theorem. □

C.2 Proof of Lemma 8.11

Let $G : \{0,1\}^\kappa \to \{0,1\}^{2\kappa}$ be a PRG. Consider the following construction.

- $\mathsf{Gen}(1^\kappa)$ takes as an input the security parameter 1^κ, samples $\tilde{\mathsf{crs}} \leftarrow \tilde{\mathsf{Gen}}(1^\kappa)$, samples $r \leftarrow \{0,1\}^{2\kappa}$, and outputs $(\tilde{\mathsf{crs}}, r)$.
- The prover $\mathcal{P}(1^\kappa, \mathsf{crs}, f, \mathbf{x})$ takes as an input the security parameter 1^κ, a CRS $\mathsf{crs} = (\tilde{\mathsf{crs}}, r)$, a circuit f and an assignment $\mathbf{x}$. Let $f_r(\mathbf{x}', \mathbf{s}')$ be the circuit that takes an assignment $\mathbf{x}'$ and a string $\mathbf{s}' \in \{0,1\}^\kappa$, and outputs 1 if either (1) $f(\mathbf{x}') = 1$, or (2) $G(\mathbf{s}') = r$. The prover computes $\tilde{\pi} \leftarrow \tilde{\mathcal{P}}(1^\kappa, \tilde{\mathsf{crs}}, f_r, (\mathbf{x}, 0^\kappa))$ and outputs $\pi = \tilde{\pi}$.
- The verifier $\mathcal{V}(1^\kappa, \mathsf{crs}, f, \pi)$ takes as an input the security parameter 1^κ, a CRS $\mathsf{crs} = (\tilde{\mathsf{crs}}, r)$, a circuit f and a proof $\pi = \tilde{\pi}$. The verifier computes the circuit f_r, and outputs $\tilde{\mathcal{V}}(1^\kappa, \tilde{\mathsf{crs}}, f_r, \tilde{\pi})$.

It is not hard to verify that completeness holds. We continue with the analysis of soundness and zero-knowledge.

Soundness. Fix a family of malicious provers $\mathcal{P}^*$, and say that $\mathcal{P}^*$ violates the soundness property with probability $\delta(\kappa)$. We observe that the probability over $r \leftarrow \{0,1\}^{2\kappa}$ that there exists s such that $G(s) = r$ is at most $2^{\kappa} \cdot 2^{-2\kappa} = 2^{-\kappa}$. Therefore, with probability $\delta(\kappa) - 2^{-\kappa}$, the adversary $\mathcal{P}^*$ outputs (f, π) so that (1) f is not satisfiable, (2) r is not in the image of G, and (3) $\tilde{\mathcal{V}}(1^{\kappa}, \tilde{\mathsf{crs}}, f_r, \pi) = 1$. Properties (1)–(2) imply that f_r is not satisfiable, and therefore, with probability $\delta(\kappa) - 2^{-\kappa}$, the adversary $\mathcal{P}^*$ violates the soundness of $\tilde{\Pi}$. We conclude that for all sufficiently large κ it holds that $\delta(\kappa) - 2^{-\kappa} \leq \epsilon_s(\kappa)$, i.e., $\delta(\kappa) \leq \epsilon_s(\kappa) + 2^{-\kappa}$, as required.

Zero knowledge. Let $\tilde{\mathsf{Sim}}$ be the (inefficient) simulator of $\tilde{\Pi}$. The efficient simulator $\mathsf{Sim}(1^{\kappa}, f)$ of Π takes the security parameter 1^{κ} and a circuit f on n input bits, samples $\tilde{\mathsf{crs}} \leftarrow \tilde{\mathsf{Gen}}(1^{\kappa})$, samples $s \leftarrow \{0,1\}^{\kappa}$, computes $r' = G(s)$, sets $\mathsf{crs} = (\tilde{\mathsf{crs}}, r')$, samples $\tilde{\pi} \leftarrow \mathcal{P}(1^{\kappa}, \tilde{\mathsf{crs}}, f_{r'}, (0^n, s))$ and outputs $(\mathsf{crs}, \tilde{\pi})$.

Consider any ensemble $\{(f_{\kappa}, \mathbf{x}_{\kappa})\}_{\kappa \in \mathbb{N}}$ of polynomially-bounded circuits f_{κ} with $f_{\kappa}(\mathbf{x}_{\kappa}) = 1$ for all $\kappa \in \mathbb{N}$. By the security of the PRG it holds that a real-world proof $(r, \tilde{\mathsf{crs}}, \mathcal{P}(1^{\kappa}, \tilde{\mathsf{crs}}, f_{\kappa,r}, (\mathbf{x}_{\kappa}, 0^{\kappa})))$ is computationally indistinguishable from $(r', \tilde{\mathsf{crs}}, \mathcal{P}(1^{\kappa}, \tilde{\mathsf{crs}}, f_{\kappa,r'}, (\mathbf{x}_{\kappa}, 0^{\kappa})))$, where $\tilde{\mathsf{crs}} \leftarrow \tilde{\mathsf{Gen}}(1^{\kappa})$, $r \leftarrow \{0,1\}^{2\kappa}$ and $r' = G(s)$ for $s \leftarrow \{0,1\}^{\kappa}$. In addition,

$$(r', \tilde{\mathsf{crs}}, \mathcal{P}(1^{\kappa}, \tilde{\mathsf{crs}}, f_{\kappa,r'}, (\mathbf{x}_{\kappa}, 0^{\kappa}))) \approx_{\epsilon_w(\kappa)} (r', \tilde{\mathsf{Sim}}(1^{\kappa}, f_{\kappa,r'})),$$

and similarly,

$$\mathsf{Sim}(1^{\kappa}, f_{\kappa}) = (r', \tilde{\mathsf{crs}}, \mathcal{P}(1^{\kappa}, f_{\kappa,r'}, (0^{n_{\kappa}}, s))) \approx_{\epsilon_w(\kappa)} (r', \tilde{\mathsf{Sim}}(1^{\kappa}, f_{\kappa,r'})),$$

where n_{κ} is the input-length of f_{κ}. The lemma follows.[14] □

C.3 Proof of Lemma 8.12

Denote by $\Pi_{\mathsf{Commit}} = (\mathsf{Gen}_{\mathsf{Commit}}, \mathsf{Commit}, \mathsf{VOpen})$ the statistically-hiding commitment scheme. Consider the following construction.

- $\mathsf{Gen}(1^{\kappa})$ takes as an input the security parameter 1^{κ}, samples $\tilde{\mathsf{crs}} \leftarrow \tilde{\mathsf{Gen}}(1^{\kappa})$, samples $\mathsf{pp} \leftarrow \mathsf{Gen}_{\mathsf{Commit}}(1^{\kappa})$, samples $(C, o) \leftarrow \mathsf{Commit}_{\mathsf{pp}}(1^{\kappa}, 0)$, and outputs $(\tilde{\mathsf{crs}}, \mathsf{pp}, C)$.
- The prover $\mathcal{P}(1^{\kappa}, \mathsf{crs}, f, \mathbf{x})$ takes as an input the security parameter 1^{κ}, a CRS $\mathsf{crs} = (\tilde{\mathsf{crs}}, \mathsf{pp}, C)$, a circuit f and an assignment $\mathbf{x}$. Let $f_{\mathsf{pp},C}(\mathbf{x}', o')$ be the circuit that takes an assignment $\mathbf{x}'$ and an opening o', and outputs 1 if either (1) $f(\mathbf{x}') = 1$, or (2) $\mathsf{VOpen}_{\mathsf{pp}}(1^{\kappa}, C, o', 1) = 1$. The prover computes $\tilde{\pi} \leftarrow \tilde{\mathcal{P}}(1^{\kappa}, \tilde{\mathsf{crs}}, f_{\mathsf{pp},C}, (\mathbf{x}, 0^{\ell}))$ where ℓ is the bit-length of an opening and outputs $\pi = \tilde{\pi}$.
- The verifier $\mathcal{V}(1^{\kappa}, \mathsf{crs}, f, \pi)$ takes as an input the security parameter 1^{κ}, a CRS $\mathsf{crs} = (\tilde{\mathsf{crs}}, \mathsf{pp}, C)$, a circuit f and a proof $\pi = \tilde{\pi}$. The verifier computes the circuit $f_{\mathsf{pp},C}$, and outputs $\tilde{\mathcal{V}}(1^{\kappa}, \tilde{\mathsf{crs}}, f_{\mathsf{pp},C}, \tilde{\pi})$.

It is not hard to verify that completeness holds. We continue with the analysis of soundness and zero-knowledge.

[14] Formally, we've showed that for every polynomial-time distinguisher, the distinguishing advantage is at most $2\epsilon_z(\kappa) + \mathsf{negl}(\kappa)$, where negl is a negligible function that might depend on the adversary. A bound of $2\epsilon_z(\kappa) + \mathsf{negl}(\kappa)$ with a global negligible function negl that is independent of the adversary, can now be obtained by the work of [Bel02].

Proof of knowledge. We let $\mathsf{Gen}'(1^\kappa)$ sample $(\tilde{\mathsf{crs}}', \tilde{s}') \leftarrow \tilde{\mathsf{Gen}}'(1^\kappa)$, $\mathsf{pp} \leftarrow \mathsf{Gen}_{\mathsf{Commit}}(1^\kappa)$ and $(C, o) \leftarrow \mathsf{Commit}_{\mathsf{pp}}(1^\kappa, 0)$ and output $\mathsf{crs}' = (\tilde{\mathsf{crs}}', \mathsf{pp}, C)$. It is not hard to verify that CRS indistinguishability follows from that of $\tilde{\Pi}$. We let Ext on input 1^κ, $\mathsf{crs}' = (\tilde{\mathsf{crs}}', \mathsf{pp}, C)$, trapdoor s, circuit-SAt instance f and proof π to execute $\tilde{\mathsf{Ext}}(1^\kappa, \tilde{\mathsf{crs}}, s', f_{\mathsf{pp},C}, \pi) = (\mathbf{x}', o')$ and output $\mathbf{x}'$.

Fix a family of polynomial-sized non-uniform malicious provers $\mathcal{P}^*$, and consider the experiment where we first sample $(\mathsf{crs}', s') \leftarrow \mathsf{Gen}'(1^\kappa)$, execute $(f, \pi) \leftarrow \mathcal{P}^*_\kappa(\mathsf{crs}')$, and compute $\mathsf{Ext}'(1^\kappa, \mathsf{crs}', s', f, \pi) = \mathbf{x}$. We assume without loss of generality that it always holds that $\mathcal{V}(1^\kappa, \mathsf{crs}', f, \pi) = 1$, since otherwise the extraction property is not violated. By the extraction of $\tilde{\Pi}$, with all but negligible probability it holds that $(\mathbf{x}', o')$ satisfies $f_{\mathsf{pp},C}$. By the binding property, with all but negligible probability it holds that $\mathsf{VOpen}_{\mathsf{pp}}(1^\kappa, C, o, 1) = 0$ and therefore it must hold that $f(\mathbf{x}') = 1$, as required.

Statistical zero-knowledge. The simulator Sim on 1^κ and f samples $\tilde{\mathsf{crs}} \leftarrow \tilde{\mathsf{Gen}}(1^\kappa)$, $\mathsf{pp} \leftarrow \mathsf{Gen}_{\mathsf{Commit}}(1^\kappa)$, and $(C, o) \leftarrow \mathsf{Commit}_{\mathsf{pp}}(1^\kappa, 1)$, and sets $\mathsf{crs} = (\tilde{\mathsf{crs}}, \mathsf{pp}, C)$. The simulator computes $\pi \leftarrow \mathcal{P}(1^\kappa, \mathsf{crs}, f_{\mathsf{pp},C}, (0^n, o))$ where n is the input length of f, and outputs (crs, π). The claim now follows by the statistical hiding property of the commitment scheme, and the statistical witness indistinguishability of $\tilde{\Pi}$. This concludes the proof of the lemma. □

C.4 Proof of Lemma 8.13

The transformation is essentially the same transformation as in [BG24, Section 4.1] from soundness to their notion of soundness*. For completeness, we continue with a formal proof. Let $(\mathsf{Gen}_{pke}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme. Consider the following construction.

- $\mathsf{Gen}(1^\kappa)$ takes as an input the security parameter 1^κ, samples $\tilde{\mathsf{crs}} \leftarrow \tilde{\mathsf{Gen}}(1^\kappa)$, samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}_{pke}(1^\kappa)$, and outputs $(\tilde{\mathsf{crs}}, \mathsf{pk})$.
- The prover $\mathcal{P}(1^\kappa, \mathsf{crs}, f, \mathbf{x})$ takes as an input the security parameter 1^κ, a CRS $\mathsf{crs} = (\tilde{\mathsf{crs}}, \mathsf{pk})$, a circuit f and an assignment $\mathbf{x}$. The prover samples an encryption $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{x})$, and let us denote the internal randomness of Enc by $\mathbf{r}$. Let $f_{\mathsf{pk},c}(\mathbf{x}', \mathbf{r}')$ be the circuit that takes an assignment $\mathbf{x}'$ and a string $\mathbf{r}'$, and outputs 1 if $f(\mathbf{x}') = 1$ and $\mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}'; \mathbf{r}') = c$, and outputs 0 otherwise. The prover computes $\tilde{\pi} \leftarrow \tilde{\mathcal{P}}(1^\kappa, \tilde{\mathsf{crs}}, f_{\mathsf{pk},c}, (\mathbf{x}, \mathbf{r}))$ and outputs $(c, \tilde{\pi})$.
- The verifier $\mathcal{V}(1^\kappa, \mathsf{crs}, f, \pi)$ takes as an input the security parameter 1^κ, a CRS $\mathsf{crs} = (\tilde{\mathsf{crs}}, \mathsf{pk})$, a circuit f and a proof $\pi = (c, \tilde{\pi})$. The verifier computes the circuit $f_{\mathsf{pk},c}$, and outputs $\tilde{\mathcal{V}}(1^\kappa, \tilde{\mathsf{crs}}, f_{\mathsf{pk},c}, \tilde{\pi})$.

For the proof of knowledge, we also define the following algorithms.

- $\mathsf{Gen}'(1^\kappa)$ takes as an input the security parameter 1^κ, samples $\tilde{\mathsf{crs}} \leftarrow \tilde{\mathsf{Gen}}(1^\kappa)$, samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}_{pke}(1^\kappa)$, and outputs $\mathsf{crs}' = (\tilde{\mathsf{crs}}, \mathsf{pk})$ and $s' = \mathsf{sk}$.
- The extractor $\mathsf{Ext}(1^\kappa, \mathsf{crs}', s', f, \pi)$ takes as an input the security parameter 1^κ, a $\mathsf{crs}' = (\tilde{\mathsf{crs}}, \mathsf{pk})$, the trapdoor $s' = \mathsf{sk}$, a circuit f and a proof $\pi = (c, \tilde{\pi})$. The extractor decodes $\mathbf{x} := \mathsf{Dec}_{\mathsf{sk}}(c)$ and outputs $\mathbf{x}$.

Proof of knowledge. First, it is not hard to see that the CRS generated by Gen' has the same distribution as the CRS generated by Gen. We say that the PKE's keys $(\mathsf{pk}, \mathsf{sk})$ are *good* if they satisfy $\Pr[\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(\mathbf{m}))] = 1$ for every string $\mathbf{m} \in \{0,1\}^*$, and we are promised that the keys are good with probability at least $1 - \mathsf{negl}(\kappa)$.

Fix any polynomial-sized non-uniform circuit family of malicious provers $\mathcal{P}^*$. Consider the extraction experiment in which we sample $\mathsf{crs}' = (\tilde{\mathsf{crs}}, \mathsf{pk})$ and $s' = \mathsf{sk}$ from $\mathsf{Gen}'(1^\kappa)$, and let $\mathcal{P}^*_\kappa(\mathsf{crs}')$ generate a circuit f and a proof π. We observe if the keys are good and the extraction property is violated, then the soundness of $\tilde{\Pi}$ is violated as well. Indeed, assume that the keys are good, and the adversary successfully generated a circuit f and a proof $\pi = (c, \tilde{\pi})$, such that $\mathsf{Dec}_{\mathsf{sk}}(c) = \mathbf{x}$ and $f(\mathbf{x}) = 0$ and $\tilde{\mathcal{V}}(1^\kappa, \tilde{\mathsf{crs}}, f_{\mathsf{pk},c}, \tilde{\pi}) = 1$. Since the keys are good, for every $\mathbf{x}' \neq \mathbf{x}$ and every $\mathbf{r}'$ it holds that $\mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}'; \mathbf{r}') \neq c$, or otherwise, when decrypting c the decryption would be $\mathbf{x} \neq \mathbf{x}'$. Now, if there is no $\mathbf{r}$ such that $\mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}; \mathbf{r}) = c$ then $f_{\mathsf{pk},c}$ is not satisfiable, and soundness is violated. Otherwise, if there exists $\mathbf{r}$ such that $\mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}; \mathbf{r}) = c$, then the only assignment $(\mathbf{x}', \mathbf{r}')$ to $f_{\mathsf{pk},c}$ that satisfies $\mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}'; \mathbf{r}') = c$ is with $\mathbf{x}' = \mathbf{x}$. But $\mathbf{x}$ does not satisfy f, and therefore $f_{\mathsf{pk},c}$ is not satisfiable and soundness is violated.

We conclude that any adversary that violates proof of knowledge with probability $\delta(\kappa)$ can be transformed into an adversary that violates the soundness of $\tilde{\Pi}$ with probability $\delta(\kappa) - \mathsf{negl}(\kappa)$. Therefore, our scheme is $(\epsilon_s(\kappa) + \mathsf{negl}(\kappa))$-proof of knowledge.

Zero knowledge. Let $\tilde{\mathsf{Sim}}$ be the (efficient) simulator of $\tilde{\Pi}$. The simulator $\mathsf{Sim}(1^\kappa, f)$ of Π takes the security parameter 1^κ and a circuit f on n input bits, samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}_{pke}(1^\kappa)$, encrypts $c_0 \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0^n)$, samples $(\tilde{\mathsf{crs}}, \tilde{\pi}) \leftarrow \tilde{\mathsf{Sim}}(1^\kappa, f_{\mathsf{pk},c_0})$, and outputs (crs, π) where $\mathsf{crs} = (\tilde{\mathsf{crs}}, \mathsf{pk})$ and $\pi = (c_0, \tilde{\pi})$.

Consider any ensemble $\{(f_\kappa, \mathbf{x}_\kappa)\}_{\kappa \in \mathbb{N}}$ of polynomially-bounded circuits f_κ with $f_\kappa(\mathbf{x}_\kappa) = 1$ for all $\kappa \in \mathbb{N}$. By the security of the PKE we obtain that the simulation $\mathsf{Sim}(1^\kappa, f_\kappa) = (\mathsf{pk}, c_0, \tilde{\mathsf{Sim}}(1^\kappa, f_{\kappa,\mathsf{pk},c_0}))$ is computationally indistinguishable from $(\mathsf{pk}, c, \tilde{\mathsf{Sim}}(1^\kappa, f_{\kappa,\mathsf{pk},c}))$ where $c = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}_\kappa)$. By the $\epsilon_z(\kappa)$-zero-knowledge of $\tilde{\Pi}$ we obtain that

$$(\mathsf{pk}, c, \tilde{\mathsf{Sim}}(1^\kappa, f_{\kappa,\mathsf{pk},c})) \approx_{\epsilon_z(\kappa)} (\mathsf{pk}, c, \tilde{\mathsf{crs}}, \tilde{\pi}),$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}_{pke}(1^\kappa)$, $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{x}_\kappa; \mathbf{r})$ where $\mathbf{r}$ is the randomness of Enc, $\tilde{\mathsf{crs}} \leftarrow \tilde{\mathsf{Gen}}(1^\kappa)$ and $\tilde{\pi} \leftarrow \tilde{\mathcal{P}}(1^\kappa, \tilde{\mathsf{crs}}, f_{\mathsf{pk},c}, (\mathbf{x}_\kappa, \mathbf{r}))$. This concludes the proof of the claim.[15] □

[15] Like in Footnote 14, a zero-knowledge error of $\epsilon_z(\kappa) + \mathsf{negl}(\kappa)$ for a global negligible function negl that is independent of the adversary, is now obtained by the work of [Bel02].