How to Construct a Leakage-Resilient (Stateless) Trusted Party

Daniel Genkin[1,2], Yuval Ishai[3,4], and Mor Weiss[5(✉)]

[1] University of Pennsylvania, Philadelphia, USA
danielg3@cis.upenn.edu
[2] University of Maryland, College Park, USA
[3] Technion, Haifa, Israel
yuvali@cs.technion.ac.il
[4] UCLA, Los Angeles, USA
[5] Northeastern University, Boston, USA
m.weiss@northeastern.onmicrosoft.com

Abstract. Trusted parties and devices are commonly used in the real world to securely perform computations on secret inputs. However, their security can often be compromised by side-channel attacks in which the adversary obtains partial leakage on intermediate computation values. This gives rise to the following natural question: *To what extent can one protect the trusted party against leakage?*

Our goal is to design a hardware device T that allows $m \geq 1$ parties to securely evaluate a function $f(x_1, \ldots, x_m)$ of their inputs by feeding T with encoded inputs that are obtained using local secret randomness. Security should hold even in the presence of an active adversary that can corrupt a subset of parties and obtain restricted leakage on the internal computations in T.

We design hardware devices T in this setting both for zero-knowledge proofs and for general multi-party computations. Our constructions can unconditionally resist either AC^0 leakage or a strong form of "only computation leaks" (OCL) leakage that captures realistic side-channel attacks, providing different tradeoffs between efficiency and security.

Keywords: Leakage-resilience · Secure multiparty computation · Algebraic manipulation detection · AMD Circuits.

1 Introduction

There is a long and successful line of work on protecting general computations against partial information leakage. Originating from the works on general secure multiparty computation (MPC) [4,11,22,37], the question has been "scaled down" to the domain of protecting circuits against local probing attacks [26] and then extended to different types of global information leakage [7–10,13,15,16,23–25,28,31,32,34].

Y. Kalai and L. Reyzin (Eds.): TCC 2017, Part II, LNCS 10678, pp. 209–244, 2017.
https://doi.org/10.1007/978-3-319-70503-3_7

Most of the works along this line consider the challenging goal of protecting computations against *continual leakage*. In a general instance of this problem, a desired ideal functionality is specified by a *stateful* circuit C, which maps the current input and state to the current output and the next state. The input and output are considered to be public whereas the state is secret. The goal is to securely realize the functionality C by a leakage-resilient randomized circuit $\hat{C}$. The circuit $\hat{C}$ is initialized with some randomized encoding $\hat{s}$ of an initial secret state s. The computation can then proceed in a virtually unlimited number of rounds, where in each round $\hat{C}$ receives an input, produces an output, and replaces the old encoding of the secret state by a fresh encoding of a new state.

The correctness goal is to ensure that $\hat{C}[\hat{s}]$ has the same input-output functionality as $C[s]$. The security goal is defined with respect to a class $\mathcal{L}$ of *leakage functions* ℓ, where each function ℓ returns some partial information on the values of the internal wires of $\hat{C}$. The adversary may adaptively choose a different function $\ell \in \mathcal{L}$ in each round. The security goal is to ensure that whatever the adversary learns by interacting with $\hat{C}[\hat{s}]$ and by additionally observing the leakage, it can simulate by interacting with $C[s]$ without obtaining any leakage.

While general solutions to the above problem are known for broad classes of leakage functions $\mathcal{L}$, they leave much to be desired. Some rely on leak-free hardware components [15,16,23,28,32]. Others make a heavy use of public-key cryptography [7,10,23,25,28] or even indistinguishability obfuscation [25]. Other issues include the need for internal fresh randomness in each round, big computational overhead that grows super-linearly with the amount of tolerable leakage, complex and subtle analysis, and poor concrete parameters. All of the above works suffer from at least some of these limitations.

In this work we take a step back, and study a simpler *stateless* variant of the problem, where both C and $\hat{C}$ are stateless circuits. The goal is to replace an ideal computation of $C(x)$ by a functionally equivalent but leakage-resilient computation $\hat{C}(\hat{x})$. Here x is a secret input which is randomly encoded into an encoded input $\hat{x}$ to protect it against leakage. Solutions for the above continuous leakage model can be easily specialized to the stateless model by considering a single round where the input is used as the initial secret state. This stateless variant of the problem has been considered before [25,26,32], but mainly as an intermediate step and not as an end goal.

Our work is motivated by the observation that this simpler setting, which is relevant to many real-world scenarios, does not only offer an opportunity to get around the limitations of previous solutions, but also poses new challenges that were not addressed before. For instance, can correctness be guaranteed even when the input encoding $\hat{x}$ is invalid, in the sense that the output corresponds to *some* valid input x? Can the solutions be extended to the case where the encoded inputs for $\hat{C}$ are contributed by several, mutually distrusting, parties? To further motivate these questions, we put them in the context of natural applications.

Protecting a trusted party. We consider the goal of protecting (stateless) trusted parties against leakage. Trusted Parties (TPs) are commonly used to perform computations that involve secret inputs. They are already widely deployed in

payment terminals and access control readers, and will be even more so in future Trusted Platform Modules. TPs have several advantages over distributed protocols for secure multiparty computation (MPC) [4,11,22,37]. First, they avoid the expensive interaction typically required by MPC protocols. Second, they are very light-weight and allow the computational complexity of the other (untrusted) parties to be independent of the complexity of the computation being performed. Finally, TPs may offer *unconditional* security against *computationally unbounded* adversaries.

An important special case which is a major focus of this work is that of a hardware implementation of zero-knowledge (ZK) proofs, a fundamental primitive for identification and a useful building block for cryptographic protocol design. Informally, a ZK hardware takes a statement and witness from a prover, and outputs the verified statement, or rej, to a verifier. While there are efficient ZK protocols without hardware (including non-interactive zero-knowledge protocols (NIZKs) [21,35], or succinct non-interactive arguments of knowledge (SNARKs) [5]), such protocols do not (and cannot) have the last two features of TP-based solutions.

A primary concern when using trusted hardware are so-called "side-channel" attacks which allow the adversary to obtain leakage on the internal computations of the device (e.g., through measuring its running time [30], power consumption [29], or the electromagnetic radiation it emits [33]). Such attacks were shown to have devastating effects on security. As discussed above, a large body of works attempted to incorporate the information obtained through such leakage into the security model, and develop schemes that are provably secure in these models. More specifically, these works have focused on designing leakage-resilient circuit compilers (LRCCs) that, informally, compile any circuit C into its leakage-resilient version $\hat{C}$, where $\hat{C}$ withstands side-channel attacks in the sense that these reveal nothing about the (properly encoded) input $\hat{x}$. However, all of the schemes obtained in these works suffer from some of the limitations discussed above. In particular, none considers the questions of invalid encodings provided by malicious parties or combining encoded inputs that originate from mutually distrusting parties. These questions arise naturally in the context of ZK and in other contexts where TPs are used.

1.1 Our Contribution

Our main goal is to study the feasibility and efficiency of protecting TPs against general classes of leakage, without leak-free hardware or trusted setup. Eliminating the leak-free hardware unconditionally [24], or under computational assumptions [13,34] has been a major research goal. However, in contrast to earlier works, we consider here the easier case of realizing a *stateless* TP in the presence of *one-time* leakage.

We model the TP as a leaky (but otherwise trusted) hardware device $\mathcal{T}$ that is used by $m \geq 1$ parties to execute a multiparty computation task. More specifically, in this setting each party locally encodes its input and feeds the encoded input into the device, that evaluates a boolean (or arithmetic) circuit

on the encoded inputs, and returns the output. This computation should preserve the secrecy of the inputs, as well as the correctness of the output, in the presence of a computationally-unbounded adversary that corrupts a subset of the parties, and additionally obtains leakage on the internals of the device. (Notice that the secrecy requirement necessitates some encoding of the inputs, otherwise we cannot protect even against a probing attack on a single bit.)

We note that the stateless hardware should be reusable on an arbitrary number of different inputs. Thus, we cannot take previous leakage-secure computation protocols that employ correlated randomness (such as the ones from [15,16]) and embed this randomness into the hardware. Indeed, we consider the internals of the hardware as being public, since any secret internal embedded values can be leaked over multiple invocations.

The model has several different variants, depending on whether the adversary is passive (i.e., only sees the inputs of corrupted parties and obtains leakage on the internals of the TP) or active (namely, it may also cause corrupted parties to provide the TP with ill-formed "encoded" inputs that may not correspond to any inputs for the original computation); whether there is a single party providing input to the TP (as in the ZK example described below) or multiple parties; whether the TP is deterministic or randomized (namely, has randomness gates that generate uniformly-random bits); and finally, whether the output of the TP is encoded or not (in the latter, one cannot protect the privacy of the output even when the adversary only obtains leakage on the internals of the TP *without* corrupting any parties, whereas in the former the outputs will remain private in this case). We focus on the variant with an active adversary, and a randomized TP with encoded outputs. We consider both the single-party and multi-party setting. In the ZK setting, we also construct deterministic TPs (at the expense of somewhat increasing the complexity of the prover and verifier).

The leakage model. We consider an extended version of the "only computation leaks" (OCL) model of Micali and Reyzin [31], also known as "OCL+" [6]. Informally, in this context, the wires of the circuit $\hat{C}$ are partitioned into a "left component" $\hat{C}_L$ and a "right component" $\hat{C}_R$. Leakage functions correspond to bounded-communication 2-party protocols between $\hat{C}_L, \hat{C}_R$, where the output of the leakage function is the transcript of the protocol when the views of $\hat{C}_L, \hat{C}_R$ consist of the internal values of the wires of these two "components". Following the terminology of Goyal et al. [25], we refer to this model as *bounded communication leakage (BCL)*. The model is formalized in the next definition.

Definition 1 (t-BCL [25]). *Let $t \in \mathbb{N}$ be a leakage bound parameter. We say that a deterministic 2-party protocol is* t-bounded *if its communication complexity is at most t. Given a t-bounded protocol Π, we define the* t-bounded-communication leakage *(t-BCL) function f_Π associated with Π, that given the views of the two parties, outputs the transcript of Π. The class $\mathcal{L}^t_{\mathrm{BCL}}$ consists of all t-BCL functions f_Π associated with t-bounded protocols Π, namely: $\mathcal{L}^t_{\mathrm{BCL}} = \{f_\Pi \ : \ \Pi \text{ is } t - \text{bounded}\}$.*

We say that a size-s circuit $\hat{C}$ is t-BCL resilient *if there exists a partition $\mathcal{P} = \{s_1, s_2\}$ of the wires of $\hat{C}$, such that the circuit resists any t-BCL function f_Π for a protocol Π that respects the partition $\mathcal{P}$.*

We note that BCL is broad enough to capture several realistic leakage attacks such as the sum of all circuit wires over the integers, as well as linear functions over the wires of the circuit. This captures several realistic attacks on hardware devices, where a single electromagnetic probe measures involuntary leakage which can be approximated by a linear function of the wires of the circuit.

1.2 Our Results

We construct TPs for both ZK proofs, and general MPC, which simultaneously achieve many of the desired features described above: they resist a wide class of leakage functions (BCL), without using any leak-free components, and are quite appealing from the perspective of asymptotic efficiency, since the complexity of the parties is *independent* of the size of the computation. Our constructions combine ideas and results from previous works on leakage-resilient circuits, with several new ideas, as discussed in Sect. 1.3.

TPs for ZK. In the context of ZK, the hardware device enables the verification of NP-statements of the form "$(x, w) \in \mathcal{R}$" for an NP-relation $\mathcal{R}$. That is, the prover provides (x, w) as input to the device, which computes the function $f(x, w) = (x, \mathcal{R}(x, w))$. Since the device is leaky, the prover is unwilling to provide its secret witness w to the device "in the clear". Instead, the prover prepares in advance a "leak-free" encoding $\hat{w}$ of w, which it stores on a small isolated device (such as a smartcard or USB drive). It then provides $(x, \hat{w})$ as input to the leaky device (e.g., by plugging in his smartcard) which outputs the public verification outcome. We say that the hardware device is an $\mathcal{L}$-secure ZK circuit if it resists leakage from $\mathcal{L}$ with negligible error. We construct $\mathcal{L}^t_{\mathrm{BCL}}$-secure ZK circuits for NP:

Theorem 1 (Leakage-secure ZK circuit). *For any leakage bound $t \in \mathbb{N}$, statistical security parameter $\sigma \in \mathbb{N}$, and length parameter $n \in \mathbb{N}$, any NP-relation $\mathcal{R} = \mathcal{R}(x, w)$ with verification circuit of size s, depth d, and n inputs has an $\mathcal{L}^t_{\mathrm{BCL}}$-secure ZK circuit $C_{\mathcal{R}}$ that outputs the outcome of verification, where $\mathcal{L}^t_{\mathrm{BCL}}$ is the family of all t-BCL functions. Moreover, to prove that $(x, w) \in \mathcal{R}$, the prover runs in time $\mathsf{poly}(t, \sigma, n, |w|)$, and $|C_{\mathcal{R}}| = \widetilde{O}(s + d(t + \sigma + n)) + \mathsf{poly}(t, \sigma, n)$.*

We also construct a variant of the ZK circuit that allows one to "trade" efficiency of the prover and verifier with the randomness used by the ZK circuit:

Theorem 2 (Deterministic leakage-secure ZK circuit). *For any leakage bound $t \in \mathbb{N}$, statistical security parameter $\sigma \in \mathbb{N}$, and length parameter $n \in \mathbb{N}$, any NP-relation $\mathcal{R} = \mathcal{R}(x, w)$ with verification circuit of size s, depth d, and n inputs has a* deterministic *$\mathcal{L}^t_{\mathrm{BCL}}$-secure ZK circuit $C_{\mathcal{R}}$. Moreover, $|C_{\mathcal{R}}| = \widetilde{O}(s + d(t + \sigma + n)) + \mathsf{poly}(t, \sigma, n)$, to prove that $(x, w) \in \mathcal{R}$, the prover runs in time $\widetilde{O}(s + d(t + \sigma + n)) + \mathsf{poly}(t, \sigma, n, |w|)$, and the verifier runs in time $\mathsf{poly}(t, \sigma, n)$.*

General MPC. We consider hardware devices that allow the evaluation of general functions in both the single-party setting, and the multiparty setting with $m \geq 2$. More specifically, we construct *m-party LRCCs* that given a circuit C that takes inputs from m parties, output a circuit $\hat{C}$ that operates on encoded inputs and outputs. Informally, we say the m-party LRCC is $(\mathcal{L}, \epsilon)$-secure if the evaluation of $\hat{C}$ guarantees (except with probability ϵ) privacy of the honest parties' inputs, and correctness of the output, in the presence of an adversary that may actively corrupt a strict subset of parties, and obtain leakage from $\mathcal{L}$ on the internals of the device. We construct m-party LRCCs that are secure against t-BCL:

Theorem 3 (Leakage-secure m-party LRCC). *For any leakage bound $t \in \mathbb{N}$, statistical security parameter $\sigma \in \mathbb{N}$, input and output length parameters $n, k \in \mathbb{N}$, and size and depth parameters $s, d \in \mathbb{N}$, any m-party function $f : (\{0,1\}^n)^m \to \{0,1\}^k$ computable by a circuit of size s and depth d has an m-party $(\mathcal{L}^t_{\mathrm{BCL}}, \epsilon)$-secure LRCC, where $\mathcal{L}^t_{\mathrm{BCL}}$ is the family of all t-BCL functions, and $\epsilon = \mathsf{negl}(\sigma)$. Moreover, the leakage-secure circuit has size $\widetilde{O}(s + d(t + \sigma \log m)) + m \cdot \mathsf{poly}(t, \sigma, \log m, k)$, its input encodings can be computed in time $\widetilde{O}(n) + \mathsf{poly}(t, \sigma, \log m, k)$, and its outputs can be decoded in time $\widetilde{O}(m \cdot k(t + \sigma \log m + k))$.*

1.3 Our Techniques

1.3.1 Leakage-Resilient Zero-Knowledge

Recall that the leaky ZK device allows a prover P to prove claims of the form "$(x, w) \in \mathcal{R}$" for some NP-relation $\mathcal{R}$. We model the device as a stateless boolean (or more generally, arithmetic) circuit C. Though C cannot be assumed to withstand leakage, using an LRCC it can be transformed into a leakage-resilient circuit $\hat{C}$. Informally, an LRCC is associated with a function class $\mathcal{L}$ (the *leakage class*), a (randomized) input encoding scheme E, and a (deterministic) output decoder $\mathsf{Dec}_{\mathsf{Out}}$. The LRCC compiles a circuit C into a (public) circuit $\hat{C}$ that emulates C over encoded inputs and outputs. $\hat{C}$ resists leakage from $\mathcal{L}$ in the sense that for any input z for C, and any $\ell \in \mathcal{L}$, the output of ℓ on the wire values of $\hat{C}$, when evaluated on $\mathsf{E}(z)$, can be efficiently simulated given only the description of C.

Our starting point in constructing leakage-resilient ZK hardware is the recent result of Goyal et al. [25], who use MPC protocols to protect computation against BCL leakage. More specifically, they design information-theoretically secure protocols in the OT-hybrid model that allow a user, aided by a pair of "honest-but-curious" servers, to compute a function of her input while preserving the privacy of the input and output even under BCL leakage on the internals of the servers. We observe that when these server programs are implemented as circuits (in particular, the OT calls are implemented by constant-sized sub-circuits), this construction gives an LRCC that resists BCL leakage.

In the context of designing leakage-resilient TPs, the main advantage of this construction over previous information-theoretically secure LRCCs that resist

similar leakage classes [15,16,32] is that [25] *does not use any leak-free components*. More specifically, these LRCCs use the leak-free components (or leak-free preprocessing in [23]) to generate "masks", which are structured random bits that are used to mask the internal computations in $\hat{C}$, thus guaranteeing leakage-resilience.

These leak-free components could be eliminated if the parties include the masks as part of their input encoding. However, this raises three issues. First, in some constructions (e.g. [15,16,32]) the number of masks is proportional to the size of $\hat{C}$, so the running time of the parties would not be independent of the computation size (which defeats the purpose of delegating most of the computation to the TP). Second, in the multi-party setting, it is not clear how to combine the masks provided by different parties into a single set of masks to be used in $\hat{C}$, such that these masks are *unknown to each one of the parties*, which is crucial for the leakage-resilience property to hold. (We show in [36] how to do so for the LRCC of [16] which resists AC^0 leakage, but this construction has the efficiency shortcomings mentioned above.) Finally, even with a single party, these constructions totally break when the party provides "ill-formed" masks (namely, masks that do not have the required structure), since correctness is guaranteed *only when the masks have the required structure*. This is not only a theoretical concern, but rather an *actual* one. To see why, consider the ZK setting. If the prover provides the masks to the device then it *has a way* of choosing (ill-formed) masks that flip the output gate, thus causing the device to accept false NP statements. Alternative "solutions" also fail: the device cannot verify that the masks provided by the prover are well-formed, since the aforementioned constructions *crucially* rely on the fact that the leakage-resilience simulator can use ill-formed masks; and the verifier cannot provide the masks, since leakage-resilience relies on the leakage function not knowing the masks.

Though using the LRCC of [25] eliminates all these issues, it has one shortcoming: its leakage-resilience simulator is *inefficient*. In the context of ZK hardware, this gives *witness-indistinguishability*, namely the guarantee that a malicious verifier that can leak on the internals of the ZK hardware cannot distinguish between executions on the same statement x with different witnesses w, w'. This falls short of our desired security guarantee that leakage reveals *no* information about the witness. (In particular, notice that if a statement x has only one witness then witness-indistinguishability provides no security.) We note that this weaker security guarantee is inherent to the construction of [25].

To achieve efficient simulation, we leverage the fact that the construction of [25] operates over encodings that resist BCL leakage. We observe that one can obtain simulation-based security if the encodings at the output of $\hat{C}$ are decoded using a circuit $\hat{C}_{\mathsf{Dec}}$ that "tolerates" BCL leakage, in the sense that such leakage on its *entire* wire values can be simulated given only (related) BCL leakage on the inputs and outputs of the circuit [7]. Indeed, the simulator can evaluate $\hat{C}$ on an *arbitrary* (*non*-satisfying) "witness" (thus generating the entire wire values of $\hat{C}$, and in particular allowing the simulator to compute any leakage on them), and then simulate leakage on the internals of $\hat{C}_{\mathsf{Dec}}$ by computing (related) leakage

on its inputs (namely, the outputs of $\hat{C}$) and output (which is $(x, 1)$). Since the outputs of $\hat{C}$ resist BCL leakage, this is indistinguishable from the leakage on the internal wires of $\hat{C}, \hat{C}_{\mathsf{Dec}}$ when $\hat{C}$ is evaluated on an actual witness. We note that the decoding circuit $\hat{C}_{\mathsf{Dec}}$ can be constructed using the LRCC of [15], which by a recent result of Bitansky et al. [8] is leakage-tolerant against BCL leakage.

Though this construction achieves efficient simulation, it is no longer sound. Indeed, soundness crucially relies on the fact that $\hat{C}_{\mathsf{Dec}}$ emulates C_{Dec} (which decodes the output of $\hat{C}$). Recall that in current LRCC constructions that offer information-theoretic security against wide leakage classes (e.g., [15,16,32]), the correctness of the computation crucially relies on the fact that the masks (which are provided as part of the input encoding) have the "correct" structure. Consequently, by providing $\hat{C}_{\mathsf{Dec}}$ with *ill-formed* masks, a malicious prover P^* can *arbitrarily* modify the functionality emulated by $\hat{C}_{\mathsf{Dec}}$, and in particular, may flip the output of $\hat{C}_{\mathsf{Dec}}$, causing the device to accept $x \notin L_{\mathcal{R}}$.[1] Recall that the device cannot verify that the masks are well-formed, since this would violate leakage-resilience.

To overcome this, we observe that when $\hat{C}_{\mathsf{Dec}}$ is generated using the LRCC of Dziembowski and Faust [15], the effect of ill-formed masks on the computation in $\hat{C}_{\mathsf{Dec}}$ is equivalent to adding a vector of fixed (but possibly different) field elements to the wires of C_{Dec}. Such attacks are called "additive attacks", and one can use *AMD circuits* [17–19] to protect against them. Informally, AMD circuits are randomized circuits that offer the best possible security under additive attacks, in the sense that the effect of every additive attack that may apply to all internal wires of the circuit can be simulated by an ideal attack that applies only to its inputs and outputs.

Thus, by replacing C_{Dec} with an AMD circuit C'_{Dec} before applying the LRCC, the effect of ill-formed encoded inputs is further restricted to an additive attack on the inputs and output of C_{Dec}. Finally, to protect the inputs and outputs of C'_{Dec} from additive attacks, we use the AMD code of [12]. (We note that encoding the inputs and outputs of C'_{Dec} using AMD codes is inherent to any AMD-based construction, otherwise a malicious prover P^* can use ill-formed encoded inputs to $\hat{C}'_{\mathsf{Dec}}$ to flip the output.) As we show in Sect. 4, the resultant construction satisfies the properties of Theorem 1. To obtain the *deterministic* circuit of Theorem 2, we have the prover provide (as part of its input encoding) the randomness used by the $\widehat{C}$ component (which was generated using the LRCC of [25]), and the verifier provides the randomness used by the AMD circuit in $\widehat{C}_{\mathsf{Dec}}$. (We note that the prover cannot provide this randomness, since the security of AMD circuits crucially relies on their randomness being *independent* of the additive attack. Therefore, if the prover provides the randomness for the AMD circuit, a malicious prover may correlate the randomness used by the AMD circuit with the additive attack, rendering the AMD circuit useless.)

[1] We note that "ill-formed" encodings do not pose a problem for *stateful* circuits (intuitively, the compiled circuit can use the secret state to overcome the influence of ill-formed masks). However, we are interested in *stateless* circuits.

1.3.2 General Leakage-Resilient Computation

Recall that the setting consists of $m \geq 1$ parties that utilize a leaky, but otherwise trusted, device to compute a joint function of their inputs; while protecting the privacy of the inputs, and the correctness of the output, against an active adversary that corrupts a subset of the parties, and may also obtain leakage on the internals of the device. More specifically, we construct *m-party LRCCs* that given a (boolean or arithmetic) circuit C with m inputs, output a circuit $\hat{C}$ that operates on encoded inputs and outputs. (Recall that encoded outputs are needed to guarantee privacy against adversaries that do not corrupt any parties.) As in other LRCCs, the circuit compiler is associated with an input encoder Enc, and an output decoder Dec (used to encode the inputs to, and the output of, $\hat{C}$, respectively).

The multiparty setting introduces an additional complication which did not arise in the ZK setting. Recall that the leakage-resilience property of $\hat{C}$ crucially relies on the fact that its internal computations are randomized using masks which are *unknown to the leakage function*. As already discussed in Sect. 1.3.1, to avoid the need for leak-free hardware we let the participating parties provide these masks. Consequently, the adversary (who also chooses the leakage function) knows the identity of the masks provided by all corrupted parties. We note that this issue occurs *even in the passive setting*, in which parties are guaranteed to honestly encode their inputs. This raises the following question: *how can we preserve the leakage-resilience property when the leakage function "knows" a subset of the masks?*

Our solution is to first replace the circuit C with a circuit C' that computes an *m-out-of-m additive secret sharing* of the output of C. We then construct the leakage-resilient version $\hat{C}'$ of C' using the LRCC of [25], which outputs encodings of the secret shares which C' computes. Then, each encoding is refreshed in a leakage-resilient manner. (This is similar to using a leakage-resilient version of the decoder in the ZK setting of Sect. 1.3.1.) More specifically, let C_{refresh} be a circuit that given an encoding of some value v outputs a fresh encoding of v. Similar to the construction of ZK circuits in Sect. 1.3.1, we replace C_{refresh} with an AMD circuit C'_{refresh} that emulates C_{refresh} but operates on AMD encodings. Finally, we compile C'_{refresh} using the LRCC of [15] into a leakage-resilient circuit $\hat{C}'_{\mathsf{refresh}}$, which (as discussed in Sect. 1.3.1) has the additional feature that ill-formed masks are detected. We use m copies of $\hat{C}'_{\mathsf{refresh}}$ to refresh the m secret shares, where the i'th copy is associated with the i'th party, who provides (as part of its input encoding) the masks needed for the computation of the i'th copy. Finally, the decoder Dec decodes the secret shares, and uses them to reconstruct the output.

Having the leakage-resilience circuit generate (encodings of) *secret-shares* of the output, instead of (an encoding of) the output itself guarantees leakage-resilience even when the adversary corrupts parties and learns the masks which they provide for the computation. At a very high level, this holds because even if the adversary learns (through the leakage, and knowledge of the masks) the *entire wire values* of the copies of $\hat{C}'_{\mathsf{refresh}}$ associated with corrupted parties, these

only reveal information about the *secret shares* which these copies operate on. Therefore, the secrecy of the secret-sharing scheme guarantees that no information is revealed about the *actual* output, or inputs, of the computation. Thus, we obtain Theorem 3. (The analysis is in fact much more complex, see Sect. 6 for the construction and its analysis.)

1.4 Open Problems

Our work leaves several interesting open problems for further research. One is that of making the TP deterministic, while minimizing the complexity of the parties. Currently, we can make the TP deterministic, but only at the expense of making the parties work as hard as the entire original computation. A natural approach is via derandomization of the LRCC of [25]. Another research direction is to obtain a better understanding of the leakage classes that can be handled in this model, and extend the results to the setting of continuous leakage with stateful circuits. Another question is that of improving the asymptotic and concrete efficiency of our constructions, by providing better underlying LRCCs, or better analysis of existing ones. These questions are interesting even in the simple setting of a single semi-honest party.

1.5 Related Work

Originating from [26], MPC techniques are commonly used as a defense against side-channel attacks (see [2,3] and references therein). However, except for the works of [14,26] (discussed below) these techniques either rely on cryptographic assumptions [13,25], or on structured randomness which is generated by leak-free hardware, and is used to mask the internal computations [6,8,15,16,23]. To eliminate the leak-free hardware, the parties can provide the structured randomness as part of their input encoding. However, since the correctness of the computation crucially relies on the randomness having the "correct" structure, this allows corrupted parties to arbitrarily modify the functionality computed by the circuit, by providing randomness that does not have the required structure.

The only exception to the above are the works of [14,26], that provide provable information-theoretic security guarantees (without relying on structured randomness) against probing attacks, and some natural types of "noisy" leakage, but fail to protect against other simple types of realistic attacks, such as the sum of a subset of wires over the integers. (For example, when an AND gate is implemented using the LRCC of [26], the sum of a subset of wires in the resultant circuit allows an adversary to distinguish between the case in which both inputs are 0, and the case in which one of them is 1.)

2 Preliminaries

Let $\mathbb{F}$ be a finite field, and Σ be a finite alphabet (i.e., a set of symbols). For a function f over Σ^n, we use $\mathsf{supp}(f)$ to denote the image of f, namely

$\mathsf{supp}(f) = \{f(x) : x \in \Sigma^n\}$. For an NP-relation $\mathcal{R} = \mathcal{R}(x, w)$, we denote $L_{\mathcal{R}} = \{x : \exists w, (x, w) \in \mathcal{R}\}$. Vectors will be denoted by boldface letters (e.g., $\mathbf{a}$). If $\mathcal{D}$ is a distribution then $X \leftarrow \mathcal{D}$, or $X \in_R \mathcal{D}$, denotes sampling X according to the distribution $\mathcal{D}$. Given two distributions X, Y, $\mathsf{SD}(X, Y)$ denotes the statistical distance between X and Y. For a natural n, $\mathsf{negl}(n)$ denotes a function that is negligible in n. For a function family $\mathcal{L}$, we sometimes use the term "leakage family $\mathcal{L}$", or "leakage class $\mathcal{L}$". In the following, n usually denotes the input length, k usually denotes the output length, d, s denote depth and size, respectively (e.g., of circuits, as defined below), and m is used to denote the number of parties.

Circuits. We consider boolean circuits C over the set $X = \{x_1, \cdots, x_n\}$ of variables. C is a directed acyclic graph whose vertices are called *gates* and whose edges are called *wires*. The wires of C are labeled with functions over X. Every gate in C of in-degree 0 has out-degree 1 and is either labeled by a variable from X and referred to as an *input gate*; or is labeled by a constant $\alpha \in \{0, 1\}$ and referred to as a const_α *gate*. Following [16], all other gates are labeled by one of the operations $\wedge, \vee, \neg, \oplus$, where $\wedge, \vee, \oplus$ vertices have fan-in 2 and fan-out 1; and $\neg$ has fan-in and fan-out 1. We write $C : \{0,1\}^n \to \{0,1\}^k$ to indicate that C is a boolean circuit with n inputs and k outputs. The *size* of a circuit C, denoted $|C|$, is the number of wires in C, together with input and output gates.

We also consider arithmetic circuits C over a finite field $\mathbb{F}$ and the set X. Similarly to the boolean case, C has input and constant gates, and all other gates are labeled by one of the following functions $+, -, \times$ which are the addition, subtraction, and multiplication operations of the field. We write $C : \mathbb{F}^n \to \mathbb{F}^k$ to indicate that C is an arithmetic circuit over $\mathbb{F}$ with n inputs and k outputs. Notice that boolean circuits can be viewed as arithmetic circuits over the binary field in a natural way. Therefore, we sometimes describe boolean circuits using the operations $+, -, \times$ instead of $\oplus, \neg, \wedge, \vee$.

Additive Attacks. Following the terminology of [17], an additive attack $\mathbf{A}$ affects the evaluation of a circuit C as follows. For every wire connecting gates a and b in C, a value specified by the attack $\mathbf{A}$ is added to the output of a and then the derived value is used for the computation of b. Similarly, for every output gate, a value specified by $\mathbf{A}$ is added to the value of this output. Note that an additive attack on C is a fixed vector of (possibly different) field elements which is independent from the inputs and internal values of C. We denote the evaluation of C under additive attack $\mathbf{A}$ by $C^{\mathbf{A}}$.

At a high level, an additively-secure implementation of a function f is a circuit which evaluates f, and guarantees the "best" possible security against additive attacks, in the sense that any additive attack on it is equivalent (up to a small statistical distance) to an additive attack on the inputs and outputs of f. Formally,

Definition 2 (Additively-secure implementation [18]). *Let $\epsilon > 0$. A randomized circuit $C : \mathbb{F}^n \to \mathbb{F}^k$ is an* ϵ-additively-secure implementation *of a function $f : \mathbb{F}^n \to \mathbb{F}^k$ if the following holds.*

- ***Completeness.*** *For every* $x \in \mathbb{F}^n$, $\Pr[C(x) = f(x)] = 1$.
- ***Additive-attack security.*** *For any additive attack* $\mathbf{A}$ *there exist* $\mathbf{a}^{\mathsf{in}} \in \mathbb{F}^n$, *and a distribution* $\mathcal{A}^{\mathsf{Out}}$ *over* $\mathbb{F}^k$, *such that for every* $\mathbf{x} \in \mathbb{F}^n$, $\mathsf{SD}(C^{\mathbf{A}}(\mathbf{x}), f(\mathbf{x} + \mathbf{a}^{\mathsf{in}}) + \mathcal{A}^{\mathsf{out}}) \leq \epsilon$.

We also consider the notion of an additively-secure circuit compiler, which is a single PPT algorithm that compiles a given circuit C into its additively-secure implementation.

Definition 3 (Additively-secure circuit compiler). *Let* $n \in \mathbb{N}$ *be an input length parameter,* $k \in \mathbb{N}$ *be an output length parameter, and* $\epsilon(n) : \mathbb{N} \to \mathbb{R}^+$. *Let* Comp *be a PPT algorithm that on input a circuit* $C : \mathbb{F}^n \to \mathbb{F}^k$, *outputs a circuit* $\hat{C}$. Comp *is an* $\epsilon(n)$-additively-secure circuit compiler over $\mathbb{F}$ *if for every circuit* $C : \mathbb{F}^n \to \mathbb{F}^k$ *that computes a function* f_C, $\hat{C}$ *is an* $\epsilon(n)$*-additively-secure implementation of* f_C.

We will need the following theorem.

Theorem 4 [19]. *Let* n *be an input length parameter, and* $\epsilon(n) : \mathbb{N} \to \mathbb{R}^+$ *be a statistical error function. Then there exists an* $\epsilon(n)$*-additively-secure circuit compiler* Comp *over* $\mathbb{F}_2$. *Moreover, on input a depth-*d *boolean circuit* $C : \{0,1\}^n \to \{0,1\}^k$, Comp *outputs a circuit* $\hat{C}$ *such that* $|\hat{C}| = |C| \cdot \mathsf{polylog}\left(|C|, \log \frac{1}{\epsilon(n)}\right) + \mathsf{poly}\left(n, k, d, \log \frac{1}{\epsilon(n)}\right)$. *Furthermore, there exists a PPT algorithm* Alg *that on input* C, $\epsilon(n)$, *and an additive attack* $\mathcal{A}$, *outputs a vector* $\mathbf{a}^{\mathsf{in}} \in \{0,1\}^n$, *and a distribution* $\mathcal{A}^{\mathsf{out}}$ *over* $\{0,1\}^k$, *such that for any* $\mathbf{x} \in \{0,1\}^n$ *it holds that* $\mathsf{SD}(\hat{C}^{\mathcal{A}}(\mathbf{x}), C(\mathbf{x} + \mathbf{a}^{\mathsf{in}}) + \mathcal{A}^{\mathsf{out}}) \leq \epsilon(n)$.

Encoding schemes. An encoding scheme E over alphabet Σ is a pair $(\mathsf{Enc}, \mathsf{Dec})$ of algorithms, where the *encoding algorithm* Enc is a PPT algorithm that given a message $x \in \Sigma^n$ outputs an encoding $\hat{x} \in \Sigma^{\hat{n}}$ for some $\hat{n} = \hat{n}(n)$; and the *decoding algorithm* Dec is a deterministic algorithm, that given an $\hat{x}$ of length $\hat{n}$ in the image of Enc, outputs an $x \in \Sigma^n$. Moreover, $\Pr[\mathsf{Dec}(\mathsf{Enc}(x)) = x] = 1$ for every $x \in \Sigma^n$. It would sometimes be convenient to explicitly describe the randomness used by Enc, in which case we think of Enc as a deterministic function $\mathsf{Enc}(x; r)$ of its input x, and random input r. Following [27], we say that a vector $\mathbf{v} \in \Sigma^{\hat{n}(n)}$ is *well-formed* if $\mathbf{v} \in \mathsf{Enc}(0^n)$.

Parameterized encoding schemes. We consider encoding schemes in which the encoding and decoding algorithms are given an additional input 1^t, which is used as a security parameter. Concretely, the encoding length depends also on t (and not only on n), i.e., $\hat{n} = \hat{n}(n, t)$, and for every t the resultant scheme is an encoding scheme (in particular, for every $x \in \Sigma^n$ *and every* $t \in \mathbb{N}$, $\Pr[\mathsf{Dec}(\mathsf{Enc}(x, 1^t), 1^t) = x] = 1$). We call such schemes *parameterized*. For $n, t \in \mathbb{N}$, a vector $\mathbf{v} \in \Sigma^{\hat{n}(n,t)}$ is *well-formed* if $\mathbf{v} \in \mathsf{Enc}(0^n, 1^t)$. Furthermore, we sometimes consider encoding schemes that take *a pair* of security parameters $1^t, 1^{t_{\mathsf{in}}}$. (t_{in} is used in cases when the encoding scheme employs an "internal"

encoding scheme, and is used in the internal scheme.) In such cases, the encoding length depends on n, t, t_{In}, and the resultant scheme should be an encoding scheme for every $t, t_{\mathsf{In}} \in \mathbb{N}$. We will usually omit the term "parameterized", and use "encoding scheme" to describe both *parameterized and non-parameterized* encoding schemes.

Next, we define leakage-indistinguishable encoding schemes.

Definition 4 (Leakage-indistinguishability of functions and encodings, [27]). *Let D, D' be finite sets, $\mathcal{L}_D = \{\ell : D \to D'\}$ be a family of leakage functions, and $\epsilon > 0$. We say that two distributions X, Y over D are* $(\mathcal{L}_D, \epsilon)$-leakage-indistinguishable, *if for any function $\ell \in \mathcal{L}_D$, $\mathsf{SD}(\ell(X), \ell(Y)) \leq \epsilon$. In case $\mathcal{L}_D$ consists of functions over a union of domains, we say that X, Y over D are $(\mathcal{L}_D, \epsilon)$-leakage-indistinguishable if $\mathsf{SD}(\ell(X), \ell(Y)) \leq \epsilon$ for every function $\ell \in \mathcal{L}$ with domain D.*

Let $\mathcal{L}$ be a family of leakage functions. We say that a randomized function $f : \Sigma^n \to \Sigma^m$ is $(\mathcal{L}, \epsilon)$-leakage-indistinguishable *if for every $x, y \in \Sigma^n$, the distributions $f(x), f(y)$ are $(\mathcal{L}, \epsilon)$-leakage-indistinguishable. We say that an encoding scheme $\mathsf{E} = (\mathsf{Enc}, \mathsf{Dec})$ is* $(\mathcal{L}, \epsilon)$-leakage-indistinguishable *if for every large enough $t \in \mathbb{N}$, $\mathsf{Enc}(\cdot, 1^t)$ is $(\mathcal{L}, \epsilon)$-leakage indistinguishable.*

Algebraic Manipulation Detection (AMD) Encoding Schemes. Informally, an AMD encoding scheme is an encoding scheme which guarantees that additive attacks on codewords are detected by the decoder (except with small probability), where the decoder outputs (in addition to the decoded output) also a flag indicating whether an additive attack was detected. Formally,

Definition 5 (AMD encoding scheme, [12,18]). *Let $\mathbb{F}$ be a finite field, $n \in \mathbb{N}$ be an input length parameter, $t \in \mathbb{N}$ be a security parameter, and $\epsilon(n, t) : \mathbb{N} \times \mathbb{N} \to \mathbb{R}^+$. An* $(n, t, \epsilon(n, t))$-algebraic manipulation detection (AMD) encoding scheme $(\mathsf{Enc}, \mathsf{Dec})$ over $\mathbb{F}$ *is an encoding scheme with the following guarantees.*

- ***Perfect completeness.*** *For every $\mathbf{x} \in \mathbb{F}^n$, $\Pr[\mathsf{Dec}(\mathsf{Enc}(\mathbf{x}, 1^t), 1^t) = (0, \mathbf{x})] = 1$.*
- ***Additive soundness.*** *For every $0^{\hat{n}(n,t)} \neq \mathbf{a} \in \mathbb{F}^{\hat{n}(n,t)}$, and every $\mathbf{x} \in \mathbb{F}^n$, $\Pr[\mathsf{Dec}(\mathsf{Enc}(\mathbf{x}, 1^t) + \mathbf{a}, 1^t) \notin \mathsf{ERR}] \leq \epsilon(n, t)$ where $\mathsf{ERR} = (\mathbb{F} \setminus \{0\}) \times \mathbb{F}^n$, and the probability is over the randomness of Enc.*

We will use the following theorem from the full version of [18].

Theorem 5 (AMD encoding scheme, [18]). *Let $\mathbb{F}$ be a finite field, and $n, t \in \mathbb{N}$. Then there exists an $\left(n, t, |\mathbb{F}|^{-t}\right)$-AMD encoding scheme $(\mathsf{Enc}, \mathsf{Dec})$ with encodings of length $\hat{n}(n, t) = O(n + t)$. Moreover, encoding and decoding of length-n inputs with parameter t can be performed by circuits of size $O(n + t)$.*

2.1 Leakage-Resilient Circuit Compilers (LRCCs)

In this section we define the notion of a leakage-resilient circuit compiler. This notion, and its variants defined in later sections, will be extensively used in this work.

Definition 6 (Circuit compiler with abort). *We say that a triplet* $(\mathsf{Comp}, \mathsf{E}, \mathsf{Dec}_{\mathsf{Out}})$ *is a* circuit compiler with abort *if:*

- $\mathsf{E} = (\mathsf{Enc}, \mathsf{Dec})$ *is an encoding scheme, where* Enc *on input* $x \in \mathbb{F}^n$*, and* $1^t, 1^{t_{\mathsf{In}}}$*, outputs a vector* $\hat{x}$ *of length* $\hat{n}$ *for some* $\hat{n} = \hat{n}(n, t, t_{\mathsf{In}})$*.*
- Comp *is a polynomial-time algorithm that given an arithmetic circuit* C *over* $\mathbb{F}$*, and* 1^t*, outputs an arithmetic circuit* $\hat{C}$*.*
- $\mathsf{Dec}_{\mathsf{Out}}$ *is a deterministic decoding algorithm associated with a length function* $\hat{n}_{\mathsf{Out}} : \mathbb{N} \to \mathbb{N}$ *that on input* $\hat{x} \in \mathbb{F}^{\hat{n}_{\mathsf{Out}}(n)}$ *outputs* $(f, x) \in \mathbb{F} \times \mathbb{F}^n$*.*

We require that $(\mathsf{Comp}, \mathsf{E}, \mathsf{Dec}_{\mathsf{Out}})$ *satisfy the following* correctness with abort *property: there exists a negligible function* $\epsilon(t) = \mathsf{negl}(t)$ *such that for any arithmetic circuit* C*, and input* x *for* C*,* $\Pr\left[\mathsf{Dec}_{\mathsf{Out}}\left(\hat{C}(\hat{x})\right) = (0, C(x))\right] \geq 1 - \epsilon(t)$*, where* $\hat{x} \leftarrow \mathsf{Enc}\left(x, 1^t, 1^{|C|}\right)$*.*

Informally, a circuit compiler is *leakage resilient* for a class $\mathcal{L}$ of functions if for every "not too large" circuit C, and every input x for C, the wire values of the compiled circuit $\hat{C}$, when evaluated on a random encoding $\hat{x}$ of x, can be simulated given only the description of C; and functions in $\mathcal{L}$ cannot distinguish between the actual and simulated wire values.

Notation 6. For a Circuit C, a function $\ell : \mathbb{F}^{|C|} \to \mathbb{F}^m$ for some natural m, and an input x for C, $[C, x]$ denotes the wire values of C when evaluated on x, and $\ell[C, x]$ denotes the output of ℓ on $[C, x]$.

Definition 7 (LRCC). *Let* $t \in \mathbb{N}$ *be a security parameter, and* $\mathbb{F}$ *be a finite field. For a function class* $\mathcal{L}$*,* $\epsilon(t) : \mathbb{N} \to \mathbb{R}^+$*, and a size function* $\mathsf{S}(n) : \mathbb{N} \to \mathbb{N}$*, we say that* $(\mathsf{Comp}, \mathsf{E}, \mathsf{Dec}_{\mathsf{Out}})$ *is an* $(\mathcal{L}, \epsilon(t), \mathsf{S}(n))$-LRCC *if there exists a PPT algorithm* Sim *such that the following holds. For all sufficiently large* t*, every arithmetic circuit* C *over* $\mathbb{F}$ *of input length* n *and size at most* $\mathsf{S}(n)$*, every* $\ell \in \mathcal{L}$ *of input length* $|\hat{C}|$*, and every* $x \in \mathbb{F}^n$*, we have* $\mathsf{SD}\left(\ell[\mathsf{Sim}(C, 1^t)], \ell\left[\hat{C}, \hat{x}\right]\right) \leq \epsilon(t)$*, where* $\hat{x} \leftarrow \mathsf{Enc}\left(x, 1^t, 1^{|C|}\right)$*.*

If the above holds with an inefficient *simulator* Sim*, then we say that* $(\mathsf{Comp}, \mathsf{E})$ *is an* $(\mathcal{L}, \epsilon(t), \mathsf{S}(n))$-relaxed LRCC*.*

2.2 Gadget-Based Leakage-Resilient Circuit Compilers

In this section we describe gadget-based LRCCs [15,16,26], which are the basis of all our constructions. We choose to describe the operation of these compilers over a finite field $\mathbb{F}$, but the description naturally adjusts to the boolean case as well. At a high level, given a circuit C, a gadget-based LRCC replaces every wire in C with a bundle of wires, which carry an encoding of the wire value, and every gate with a sub-circuit that emulates the operation of the gate on encoded inputs. More specifically:

Gadgets. A bundle is a sequence of field elements, encoding a field element according to some encoding scheme E; and a gadget is a circuit which operates on bundles and emulates the operation of the corresponding gate in C.

A gadget has both standard inputs, that represent the wires in the original circuit, and masking inputs (so-called "masks"), that are used to achieve privacy. More formally, a gadget emulates a specific boolean or arithmetic operation on the standard inputs, and outputs a bundle encoding the correct output. Every gadget G is associated with a set M_G of "well-formed" masking input bundles (e.g., in the LRCC of [16], M_G consists of sets of 0-encodings). For every standard input x, on input a bundle $\mathbf{x}$ encoding x, and *any* masking input bundles $\mathsf{m} \in M_G$, the output of the gadget G should be consistent with the operation on x. For example, if G computes multiplication, then for every standard input $x = (x_1, x_2)$, for every bundle encoding $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$ of x according to E, and for every masking input bundles $\mathsf{m} \in M_G$, $G(\mathbf{x}, \mathsf{m})$ is a bundle encoding $x_1 \times x_2$ according to E. Because the encoding schemes we use have the property that the encoding function is onto its range, we may think of the masking input bundles m as encoding some set mask of values. The internal computations in the gadget will remain private as long as its masking input bundles are a uniformly random encoding of mask, *regardless* of the actual value of mask.

Gadget-based LRCCs. In our constructions, the compiled circuit $\hat{C}$ is obtained from a circuit C by replacing every wire with a bundle, and every gate with the corresponding gadget. Recall that the gadgets also have masking inputs (which in previous works [15,16] were generated by leak-free hardware). These are provided as part of the encoded input of $\hat{C}$, in the following way. $\mathsf{E} = (\mathsf{Enc}, \mathsf{Dec})$ uses an "inner" encoding scheme $\mathsf{E}^{\mathsf{In}} = (\mathsf{Enc}^{\mathsf{In}}, \mathsf{Dec}^{\mathsf{In}})$, where Enc uses $\mathsf{Enc}^{\mathsf{In}}$ to encode the inputs of C, concatenated with $0^{t_{\mathsf{In}}}$ for a "sufficiently large" t_{In} (these 0-encodings will be the masking inputs of the gadgets, that are used to achieve privacy); and Dec uses $\mathsf{Dec}^{\mathsf{In}}$ to decode its input, and discards the last t_{In} symbols.

3 LRCCs Used in this Work

In this section we review the various LRCC constructions used in this work.

3.1 The LRCC of [25]

We use a slight modification of the LRCC of Goyal et al. [25], which we describe in this section. Their construction uses *small-bias* encodings over $\mathbb{F}_2$, namely encodings for which linear distinguishers obtain only a small distinguishing advantage between encodings of 0 and 1. Formally:

Definition 8 (Small-bias encoding schemes). *Let $\epsilon \in (0,1)$, and $(\mathsf{Enc}, \mathsf{Dec})$ be an encoding scheme over $\mathbb{F}_2$ with encodings of length $\hat{n}$. We say that $(\mathsf{Enc}, \mathsf{Dec})$ is ϵ-*biased *if for every $x \in \mathbb{F}_2$, and every $\emptyset \neq S \subseteq [\hat{n}]$, $|\Pr[P_S(\mathsf{Enc}(x)) = 1] - \Pr[P_S(\mathsf{Enc}(x)) = 0]| \leq \epsilon$, where $P_S(z) = \oplus_{i \in S} z_i$, and the probability is over the randomness of Enc.*

At a high level, given a circuit C (which, without loss of generality, contains only NAND gates), its leakage-resilient version is constructed in three steps: first, C is compiled into a *parity resilient* circuit $C_{\oplus}$, which emulates the operation of C on small-bias encodings of its inputs, and resists leakage from the class of all parity function (namely, all functions that output the parity of a subset of wires). $C_{\oplus}$ is constructed using a single constant-size gadget $\mathcal{G}$ that operates over the small-bias encoding. Second, a GMW-style 2-party protocol π is constructed, which emulates $C_{\oplus}$ (gate-by-gate) on additive secret shares of the input, and outputs additive secret shares of the output. π uses an oracle to the functionality computed by the gadget $\mathcal{G}$. In the final step, each oracle call to $\mathcal{G}$ is replaced with a constant number of OT calls, and the resultant 2-party protocol is converted into a boolean circuit, in which the OT calls are implemented using a constant number of gates.[2] The resultant circuit C' operates on encoded inputs, and returns encoded outputs. More specifically, the encoding scheme first encodes each input bit using the small-bias encoding, then additively secret shares these encodings into two shares.

The reason we need to modify the compiler is the small-bias encoding it uses. The LRCC can use *any* small-bias encoding, and [25] construct a robust gadget $\mathcal{G}$, that can emulate *any* constant-sized boolean function, over inputs and outputs encoded according to *any* constant-sized small-bias encoding (the inputs and outputs may actually be encoded using different encoding schemes). However, the specific encoding used in [25] is insufficient for our needs. More specifically, we need an encoding scheme $\Big(\mathsf{Enc} : \{0,1\} \times \{0,1\}^c \to \{0,1\}^{c'}, \mathsf{Dec} : \{0,1\}^{c'} \to \{0,1\}^2\Big)$ (for some natural constants c, c')[3] satisfying the following two properties for some constant $\epsilon > 0$.

- **Property (1):** $(\mathsf{Enc}, \mathsf{Dec})$ is ϵ-biased, and $|\mathsf{supp}(\mathsf{Enc}(0;\cdot))| = |\mathsf{supp}(\mathsf{Enc}(1;\cdot))|$.
- **Property (2):** For every $\mathbf{0} \neq \mathbf{A} \in \{0,1\}^{c'}$, and every $b \in \{0,1\}$, $\Pr_{r \in_R \{0,1\}^c}[\mathsf{Enc}(b;r) \oplus \mathbf{A} \in \mathsf{supp}(\mathsf{Enc}(1 \oplus b;\cdot))] \leq \epsilon$.

The first property is needed for the leakage-resilience property of the LRCC of [25]. The second property implies that with constant probability, additive attacks on encodings are "harmless", in the sense that they either do not change the encoded value, or result in an invalid encoding. The reason that the second property is needed will become clear in Sect. 4.1.

Since the encoding scheme used in [25] does not possess property (2), we replace it with an encoding that does.[4] As noted in [25], a probabilistic argument implies that for a large enough constant c, and $c' = 2c$, most encoding

[2] We note that the conversion from protocol to circuit is not explicitly described in [25].

[3] Dec returns a pair of bits of which one is a flag indicating whether decoding failed. This is necessary since for $c' > c + 1$, not all possible inputs to Dec are valid encoding.

[4] To improve efficiency of our construction by a factor of 2, one could use the encoding of [25] (in which $c' = c + 1$) throughout the circuit, and only use our new encoding for the outputs of the circuit. However, to simplify the construction we choose to use the same encoding throughout the circuit.

schemes with a 1:1 Enc satisfy property (1). A similar argument shows that most encoding schemes posses property (2). Therefore, there exists an encoding scheme $\left(\mathsf{Enc}^{\oplus} : \{0,1\} \times \{0,1\}^c \to \{0,1\}^{2c}, \mathsf{Dec}^{\oplus} : \{0,1\}^{2c} \to \{0,1\}^2\right)$ with both properties. (Moreover, one can find an explicit description of this scheme, since c is constant.) Since $\mathcal{G}$ is a generic gadget, that can be used to emulate any function over any encoding, we can replace the encoding scheme of [25] with $\left(\mathsf{Enc}^{\oplus}, \mathsf{Dec}^{\oplus}\right)$.

We are now ready to define the encoding used by the LRCC of [25].

Construction 1. *The encoding scheme* $\left(\mathsf{Enc}^{\mathrm{GIMSS}}, \mathsf{Dec}^{\mathrm{GIMSS}}\right)$ *over* $\mathbb{F}_2$ *is defined as follows:*

- *for every* $x \in \mathbb{F}_2$, $\mathsf{Enc}^{\mathrm{GIMSS}}\left(x, 1^t\right)$*:*
  - *Generates* $x^1, \cdots, x^t \leftarrow \mathsf{Enc}^{\oplus}(x)$.
  - *Picks* $\boldsymbol{x}^L, \boldsymbol{x}^R \in \mathbb{F}_2^{2ct}$ *uniformly at random subject to the constraint that* $\boldsymbol{x}^L \oplus \boldsymbol{x}^R = \left(x^1, \cdots, x^t\right)$.
- $\mathsf{Dec}^{\mathrm{GIMSS}} : \mathbb{F}_2^{2ct} \times \mathbb{F}_2^{2ct} \to \mathbb{F}_2^2$*, on input* $\left(\boldsymbol{x}^L, \boldsymbol{x}^R\right)$ *operates as follows:*
  - *Computes* $\boldsymbol{x} = \boldsymbol{x}^L \oplus \boldsymbol{x}^R$*, and denotes* $\boldsymbol{x} = \left(x^1, \cdots, x^t\right)$*. (Intuitively,* $\boldsymbol{x}^L, \boldsymbol{x}^R$ *are interpreted as random secret shares of* $\boldsymbol{x}$*, and* $\boldsymbol{x}$ *consists of* t *copies of encodings, according to* $\mathsf{Enc}^{\oplus}$*, of a bit* b*.)*
  - *For every* $1 \leq i \leq t$*, let* $(f_i, x_i) = \mathsf{Dec}^{\oplus}\left(x^i\right)$*. (This step decodes each of the* t *copies of* b*.)*
  - *If there exist* $1 \leq i_1, i_2 \leq t$ *such that* $f_{i_1} \neq 0$*, or* $x_{i_1} \neq x_{i_2}$*, then sets* $f = 1$*. Otherwise, sets* $f = 0$*. (This step checks that all copies of* b *are consistent, and that no flag is set, otherwise the decoder sets a flag* f*.)*
  - *Outputs* $\left(f, x^1\right)$.

We will need the fact that every additive attack on encodings generated by Construction 1 is either "harmless" (in the sense that it does not change the encoded value), or causes a decoding failure. This is formalized in the next lemma.

Lemma 1. *Let* $t \in \mathbb{N}$ *be a security parameter. Then for every* $\mathbf{0} \neq \mathbf{A} \in \mathbb{F}_2^{4ct}$*, and for every* $x \in \mathbb{F}_2$*,*

$$\Pr\left[\mathsf{Dec}^{\mathrm{GIMSS}}\left(\mathsf{Enc}^{\mathrm{GIMSS}}\left(x, 1^t\right) + \mathbf{A}\right) \notin \{(0, x), \mathsf{ERR}\}\right] = \mathsf{negl}(t).$$

Proof. Let $\mathbf{0} \neq \mathbf{A} = \left(\mathbf{A}^L, \mathbf{A}^R\right) \in \mathbb{F}_2^{2ct} \times \mathbb{F}_2^{2ct}$, and let $\left(\mathbf{x}^L, \mathbf{x}^R\right) \leftarrow \mathsf{Enc}^{\mathrm{GIMSS}}\left(x, 1^t\right)$. Then on input $\left(\mathbf{y}^L, \mathbf{y}^R\right) = \left(\mathbf{x}^L, \mathbf{x}^R\right) + \left(\mathbf{A}^L, \mathbf{A}^R\right)$, the decoder $\mathsf{Dec}^{\mathrm{GIMSS}}$ first computes

$$\mathbf{x}' = \left(x^{1\prime}, \cdots, x^{t\prime}\right) = \mathbf{y}^L \oplus \mathbf{y}^R = \mathbf{x}^L \oplus \mathbf{x}^R \oplus \mathbf{A}^L \oplus \mathbf{A}^R$$

and then for every $1 \leq i \leq t$, computes $\left(f_i, x_i'\right) \leftarrow \mathsf{Dec}^{\oplus}\left(x^i, 1^t\right)$. We consider two possible cases.

First, if $\mathbf{A}^L \oplus \mathbf{A}^R = \mathbf{0}$, then $\mathbf{x}' = \mathbf{x}^L \oplus \mathbf{x}^R$, namely the additive attack cancels out, and so the output of $\mathsf{Dec}^{\mathrm{GIMSS}}$ would be $(0, x)$ (with probability 1) by the correctness of the scheme.

Second, assume that $\mathbf{A}^L \oplus \mathbf{A}^R \neq \mathbf{0}$ and $\mathsf{Dec}^{\mathrm{GIMSS}}(\mathbf{x} \oplus \mathbf{A}, 1^t) \neq (0, x)$. We show that in this case $\mathsf{Dec}^{\mathrm{GIMSS}}$ outputs ERR except with negligible probability. Recall that $\mathsf{Enc}^{\oplus}$ has the property that for every $\mathbf{0} \neq \mathbf{A}'$, and every $z \in \mathbb{F}$, $\Pr\left[\mathsf{Enc}^{\oplus}(z) \oplus \mathbf{A}' \in \mathsf{supp}\left(\mathsf{Enc}^{\oplus}(\bar{z})\right)\right] \leq \epsilon$ for some constant $\epsilon \in (0,1)$, where the probability is over the randomness used by $\mathsf{Enc}^{\oplus}$ to generate the encoding. Consequently, for every $1 \leq i \leq t$, $\Pr\left[\mathsf{Dec}^{\oplus}\left(x^{i\prime}\right) = (0, \bar{x})\right] \leq \epsilon$. Since $\mathsf{Dec}^{\mathrm{GIMSS}}$ outputs $(0, \bar{x})$ only if all $x^{i\prime}$ decoded to $\bar{x}$, and each of these t copies was generated using fresh, independent randomness in $\mathsf{Enc}^{\oplus}$, this happens only with probability $\epsilon^t = \mathsf{negl}(t)$.

The final modification we need is in the gadget $\mathcal{G}$. Notice that unlike the semi-honest setting considered in [25], in our setting *the parties* provide the inputs to the leakage-resilient circuit, where a malicious party may provide inputs that are *not* properly encoded, and therefore do not correspond to *any* input for the original circuit. (We note that the inputs are the only encodings that may be invalid, since $\mathcal{G}$ is guaranteed to always output valid encodings.) To guarantee correctness of the computation even in this case, the encoded inputs should induce inputs to the original circuit. Therefore, we have $\mathcal{G}$ interpret invalid encodings as encoding the all-zeros string. More specifically, given encodings $\hat{x}, \hat{y}$, $\mathcal{G}$ operates as follows: decodes $\hat{x}, \hat{y}$ to obtain x, y, where if decoding failed then x, y are set to the all-zero strings; computes $z = \mathrm{NAND}(x, y)$; and outputs a fresh encoding of z.

Combining the aforementioned modifications, we have the following.

Construction 2 (LRCC, [25]). *Let $c \in \mathbb{N}$ and $\epsilon \in (0,1)$ be constants, $t, t_{\mathsf{In}} \in \mathbb{N}$ be security parameters, and $n \in \mathbb{N}$ be an input length parameter. Let $\left(\mathsf{Enc}^{\oplus} : \mathbb{F}_2 \times \mathbb{F}_2^c \to \mathbb{F}_2^{2c}, \mathsf{Dec}^{\oplus} : \mathbb{F}_2^{2c} \to \mathbb{F}_2\right)$ be an encoding scheme satisfying properties (1) and (2) described above. (We also use $\mathsf{Enc}^{\oplus}, \mathsf{Dec}^{\oplus}$ to denote the natural extension of encoding and decoding to bit strings, where every bit is encoded or decoded separately.) The relaxed LRCC with abort $\left(\mathsf{Comp}^{\mathrm{GIMSS}}, \mathsf{E}_{\mathsf{In}}^{\mathrm{GIMSS}}, \mathsf{Dec}_{\mathsf{Out}}^{\mathrm{GIMSS}}\right)$ is defined as follows.*

- *The input encoding scheme $\mathsf{E}_{\mathsf{In}}^{\mathrm{GIMSS}} = \left(\mathsf{Enc}_{\mathsf{In}}^{\mathrm{GIMSS}}, \mathsf{Dec}_{\mathsf{In}}^{\mathrm{GIMSS}}\right)$ is defined as follows:*
  - *for every $x \in \mathbb{F}_2$, $\mathsf{Enc}_{\mathsf{In}}^{\mathrm{GIMSS}}(x, 1^{t_{\mathsf{In}}}) = \left(\boldsymbol{x}^L, \boldsymbol{x}^R, \boldsymbol{r}\right)$ where $\boldsymbol{x}^L, \boldsymbol{x}^R$ are a random additive secret sharing of $\mathsf{Enc}^{\oplus}(x)$, and $r \in_R \mathbb{F}_2^{t_{\mathsf{In}}}$.*
  - *$\mathsf{Dec}_{\mathsf{In}}^{\mathrm{GIMSS}}\left(\left(\left(\boldsymbol{x}^L, \boldsymbol{x}^R\right), \boldsymbol{r}\right), 1^{t_{\mathsf{In}}}\right)$ computes $(f, x) = \mathsf{Dec}^{\oplus}\left(\boldsymbol{x}^L + \boldsymbol{x}^R\right)$, and outputs x.*
- *The output decoding algorithm $\mathsf{Dec}_{\mathsf{Out}}^{\mathrm{GIMSS}} : \mathbb{F}_2^{n \cdot t \cdot 2c} \times \mathbb{F}_2^{n \cdot t \cdot 2c} \to \mathbb{F}_2^{n+1}$, on input $\left(\boldsymbol{x}^L, \boldsymbol{x}^R\right) = \left(\left(\boldsymbol{x}_1^L, \cdots, \boldsymbol{x}_n^L\right), \left(\boldsymbol{x}_1^R, \cdots, \boldsymbol{x}_n^R\right)\right)$ operates as follows:*
  - *For every $1 \leq i \leq n$, computes $(f_i, x_i) = \mathsf{Dec}^{\mathrm{GIMSS}}\left(\left(\boldsymbol{x}_i^L, \boldsymbol{x}_i^R\right), 1^t\right)$ (where $\mathsf{Dec}^{\mathrm{GIMSS}}$ is the decoder from Construction 1).*
  - *If there exist $1 \leq i \leq n$ such that $f_i \neq 0$, outputs $(1, 0^n)$. Otherwise, outputs $(f, x_1, \cdots, x_n)$.*

– *Let $r \in \mathbb{N}$ denote the number of random inputs used by each gadget $\mathcal{G}$. Then* $\mathsf{Comp}^{\mathrm{GIMSS}}$*, on input 1^t and a circuit $C : \mathbb{F}^n \to \mathbb{F}^k$ containing s NAND gates, outputs a circuit $C^{\mathrm{GIMSS}} : \mathbb{F}_2^{4c \cdot n} \times \mathbb{F}_2^{r(s+t \cdot k)} \to \mathbb{F}_2^{4c \cdot k \cdot t}$ generated as follows:*
  - *Let $C' : \mathbb{F}_2^{2c \cdot n} \times \mathbb{F}_2^{r \cdot s} \to \mathbb{F}_2^{2c \cdot k}$ denote the circuit in which every gate of C is replaced with the gadget $\mathcal{G}$ of [25] that emulates a NAND gate over encodings generated by $\mathsf{Enc}^{\oplus}$. The random inputs used by the gadgets in C' are taken from the second input to C' (each random input is used only once).*
  - *Let $C'' : \mathbb{F}_2^{2c \cdot n} \times \mathbb{F}_2^{r(s+t \cdot k)} \to \mathbb{F}_2^{2c \cdot k \cdot t}$ denote the circuit obtained from C' by adding after each output gadget of C' (namely each gadget whose output is an output of C') t gadgets $\mathcal{G}$ emulating the identity function. As in C', the random inputs used by the gadgets in C'' are taken from the second input to C''. (This step encodes each output bit using the repetition code.)*[5]
  - *Let π denote a 2-party GMW-style protocol in the OT-hybrid model which emulates C'' gadget-by-gadget on inputs encoded according to* $\mathsf{Enc}^{\mathrm{GIMSS}}$ *(i.e., on additive shares of encodings according to $\mathsf{Enc}^{\oplus}$). Then C^{GIMSS} is the circuit obtained from π by implementing the programs of the parties as a circuit, where each OT call with inputs $(x_0, x_1), b$ is implemented using the following constant-sized circuit: $\mathsf{OT}((x_0, x_1), b) = (x_0 \wedge \bar{b}) \oplus (x_1 \wedge b)$. (The wires of this circuit are divided between the parties as follows: the input wires x_0, x_1 are assigned to the OT sender; whereas the wires corresponding to $b, \bar{b}$, the outputs of the $\wedge$ gates, and the output of the $\oplus$ gate, are assigned to the OT receiver.*[6]*)*

Goyal et al. [25] show that Construction 2 resists BCL (Definition 1):

Theorem 7 (Implicit in [25]). *For every leakage-bound $t \in \mathbb{N}$, input and output lengths $n, k \in \mathbb{N}$, and size bound $s \in \mathbb{N}$, there exists an $(\mathcal{L}^t_{\mathrm{BCL}}, 2^{-t}, s)$-relaxed LRCC with abort, where $\mathcal{L}^t_{\mathrm{BCL}}$ is the family of all t-BCL functions. Moreover, on input a size-s, depth d circuit $C : \{0,1\}^n \to \{0,1\}^k$, the leakage-resilient circuit C^{GIMSS} has size $\widetilde{O}(s + td + t^2)$, the input encoder $\mathsf{Enc}^{\mathrm{GIMSS}}_{\mathsf{In}}$ can be implemented by a circuit of size $\widetilde{O}(n + t)$, and the output decoder $\mathsf{Dec}^{\mathrm{GIMSS}}_{\mathsf{Out}}$ can be implemented by a circuit of size $\widetilde{O}(t^2 + tk)$.*[7]

[5] This step, which we add to the LRCC of [25], is used to reduce the decoding error when the LRCC is used to construct leakage-secure ZK circuits in Sect. 4.1. We note that this modification preserves the parity-resilience property since it is equivalent to duplicating each output of C t times before transforming it into C'.

[6] Notice that this division of the wires preserves the leakage-resilience guarantee of [25]. Indeed, in [25] the view of the OT sender contains the input wires x_0, x_1, whereas the view of the OT receiver contains the input wire b and the output of the OT (i.e., the output of the $\oplus$ gate). Notice that $\bar{b}$ and the outputs of the $\wedge$ gates are computable from b and the OT output, so the view of the OT receiver contains exactly the same information in [25] and in our implementation of their protocol.

[7] The output decoder in the original construction of [25] has size $\widetilde{O}(t + k)$, the decoder of Construction 2 is larger due to the modified encoding we use, which replaces each encoded output bit with t copies.

3.2 The Leakage-Tolerant Circuit-Compiler of [15]

In this section we describe the Leakage-Tolerant Circuit-Compiler (LTCC) obtained from [15] through the transformation of [8]. Informally, the LRCC of Dziembowski and Faust [15], denoted DF-LRCC, is a gadget-based LRCC which uses the inner-product encoding scheme that encodes a value x as a pair of vectors whose inner-product is x:

Definition 9 (Inner product encoding scheme). *Let $\mathbb{F}$ be a finite field, and $n \in \mathbb{N}$ be an input length parameter. The* inner product encoding scheme $\mathsf{E}_{\mathsf{IP}} = (\mathsf{Enc}_{\mathsf{IP}}, \mathsf{Dec}_{\mathsf{IP}})$ *over $\mathbb{F}$ is a parameterized encoding scheme defined as follows:*

- *For every input $x = (x_1, \cdots, x_n) \in \mathbb{F}^n$, and security parameter $t \in \mathbb{N}$, $\mathsf{Enc}_{\mathsf{IP}}(x, 1^t) = \left(\left(\boldsymbol{y}_1^L, \boldsymbol{y}_1^R\right), \cdots, \left(\boldsymbol{y}_n^L, \boldsymbol{y}_n^R\right)\right)$, where for every $1 \leq i \leq n$, $\boldsymbol{y}_i^L, \boldsymbol{y}_i^R$ are random in $\left(\mathbb{F} \setminus \{0\}\right)^t$ subject to the constraint that $\langle \boldsymbol{y}_i^L, \boldsymbol{y}_i^R \rangle = x_i$.*
- *For every $t \in \mathbb{N}$, and every $\left(\left(\boldsymbol{y}_1^L, \boldsymbol{y}_1^R\right), \cdots, \left(\boldsymbol{y}_n^L, \boldsymbol{y}_n^R\right)\right) \in \mathbb{F}^{2nt}$, $\mathsf{Dec}_{\mathsf{IP}}\left(\left(\boldsymbol{y}_1^L, \boldsymbol{y}_1^R\right), \cdots, \left(\boldsymbol{y}_n^L, \boldsymbol{y}_n^R\right)\right) = \left(\langle \boldsymbol{y}_1^L, \boldsymbol{y}_1^R \rangle, \cdots, \langle \boldsymbol{y}_n^L, \boldsymbol{y}_n^R \rangle\right)$.*

More specifically, the DF-LRCC is an LRCC variant in which the compiled circuit takes un-encoded inputs, as well as masking inputs that are used in the gadgets. The construction uses 4 gadgets: a refresh gadget which emulates the identity function, and is used to generate fresh encodings of the wires; a *generalized-multiplication* gadget which emulates the function $f_c(x, y) = c - x \times y$, for a constant $c \in \mathbb{F}$; a *multiplication by a constant* gadget that emulates the function $f_c(x) = c \times x$, for a constant $c \in \mathbb{F}$; and an *addition by a constant* gadget that emulates the function $f_c(x) = c + x$, for a constant $c \in \mathbb{F}$. (The field operations $\times, +, -$ can be implemented using a constant number of these gadgets.) For completeness, these gadgets are described in Appendix A. We will only need the following property of these gadgets: the effect of evaluating a gadget with ill-formed masking inputs is equivalent to an additive attack on the gate that the gadget emulates (this is formalized in Lemma 3).

As explained in Sect. 1.3.1, we use a leakage-tolerant variant of the DF-LRCC. Roughly speaking, a leakage-tolerant circuit operates on un-encoded inputs and outputs (the input encoding function simply returns the inputs, concatenated with masking inputs), where any leakage on the computation can be simulated by related leakage *on the inputs and outputs alone*. (Leakage on the inputs and outputs is unavoidable since these are provided to the circuit "in the clear".) Formally,

Definition 10 (LTCC (for BCL)). *Let $t, \epsilon(t), \mathsf{S}(n)$ be as in Definition 7, let $n, k \in \mathbb{N}$ be input and output length parameters (respectively), and let $\mathcal{L}_{\mathrm{BCL}}^t$ be the family of t-BCL functions. We say that a pair $(\mathsf{Comp}, \mathsf{E})$ is an $(\mathcal{L}_{\mathrm{BCL}}^t, \epsilon(t), \mathsf{S}(n))$-*leakage-tolerant circuit-compiler (LTCC) *if $\mathsf{Comp}, \mathsf{E}$ have the syntax of Definition 6, and satisfy the following properties for some negligible function $\epsilon(t) = \mathsf{negl}(t)$:*

- ***Correctness.*** *For any arithmetic circuit C, and input x for C, $\Pr\left[\hat{C}(\hat{x}) = C(x)\right] \geq 1 - \epsilon(t)$, where $\hat{x} \leftarrow \mathsf{Enc}\left(x, 1^t, 1^{|C|}\right)$.*

– ***(Oblivious) leakage-tolerance.*** *There exists a partition* $\mathcal{P} = ((n_1, n_2), (k_1, k_2))$ *of input and output lengths, and a PPT algorithm* Sim *such that the following holds for all sufficiently large* $t \in \mathbb{N}$*, all* $n, k \in \mathbb{N}$*, every arithmetic circuit* $C : \mathbb{F}^n \to \mathbb{F}^k$ *of size at most* $\mathsf{S}(n)$*, and every* $\ell \in \mathcal{L}^t_{\mathrm{BCL}}$ *of input length* $|\hat{C}|$*.* Sim *is given* C*, and outputs a* view translation circuit $\mathcal{T} = (\mathcal{T}_1, \mathcal{T}_2)$ *such that for every* $(x_1, x_2) \in \mathbb{F}^{n_1} \times \mathbb{F}^{n_2}$*,*

$$\mathsf{SD}\left(\ell\left(\mathcal{T}_1\left(x_1, C\left(x_1, x_2\right)_1\right), \mathcal{T}_2\left(x_2, C\left(x_1, x_2\right)_2\right)\right), \ell\left[\hat{C}, \left(\hat{x}_1, \hat{x}_2\right)\right]\right) \leq \epsilon\left(t\right)$$

where $C\left(x_1, x_2\right) = \left(C\left(x_1, x_2\right)_1, C\left(x_1, x_2\right)_2\right) \in \mathbb{F}^{k_1} \times \mathbb{F}^{k_2}$*.*

We use a recent result of Bitansky et al. [8], that show a general transformation from LRCCs with a strong simulation guarantee against OCL, to LTCCs. Recently, Dachman-Soled et al. [13] observed that the DF-LRCC has this strong simulation property, namely the transformation can be applied directly to the DF-LRCC.[8] The final LTCC will use the following circuit $C^{\mathrm{LR-DF}}$:

Definition 11. *Let* $t \in \mathbb{N}$ *be a security parameter, and let* $r = r\left(t\right)$ *denote the maximal length of masking inputs used by a gadget of Construction 6. For an arithmetic circuit* $C : \mathbb{F}^n \to \mathbb{F}^k$ *containing* $+$ *and* $\times$ *gates, defined the circuit* $C^{\mathrm{LR-DF}} : \mathbb{F}^{n+r(t)\cdot(n+|C|)} \to \mathbb{F}^k$ *as follows:*

– *The input* $\left(x = \left(x_1, \cdots, x_n\right), \boldsymbol{m}\right) \in \mathbb{F}^n \times \left(\mathsf{supp}\left(\mathsf{Enc}^{\mathsf{In}}_{\mathrm{DF}}\left(0, 1^t\right)\right)\right)^{|C|+n}$ *of* $C^{\mathrm{LR-DF}}$ *is interpreted as an input* x *for* C*, and a collection* $\boldsymbol{m}$ *of masking inputs for gadgets.*
– *Every gate of* C *is replaced with the corresponding gadget (as defined in Construction 6), and gadgets corresponding to output gates are followed by decoding sub-circuits (computing the decoding algorithm* $\mathsf{Dec}_{\mathsf{IP}}$ *of the inner product encoding of Definition 9). The masking inputs used in the gadgets are taken from* $\boldsymbol{m}$ *(every masking input in* $\boldsymbol{m}$ *is used at most once).*
– *Following each input gate* x_i*, an encoding sub-circuit (with some fixed, arbitrary randomness hard-wired into it) is added, computing the inner-product encoding of* x_i*.*
– *A refresh gadget is added following every encoding sub-circuit, where the masking inputs used in the gadgets are taken from* $\boldsymbol{m}$*.*

We now describe the LTCC of [15]. To simplify the notations and constructions, we define the LTCC only for circuits operating on pairs of inputs.

Construction 3 (LTCC, [15] and [8]). *Let* $t, t_{\mathsf{In}} \in \mathbb{N}$*, and* $n \in \mathbb{N}$ *be an input length parameter. Let* $\mathsf{S} : \mathbb{N}^4 \to \mathbb{N}$ *be a length function whose value is set below. The LTCC* $\left(\mathsf{Comp}^{\mathrm{DF}}, \mathsf{E}^{\mathrm{DF}}\right)$ *is defined as follows:*

[8] We note that though Bitansky et al. [8] construct leakage-tolerant circuits based on the DF-LRCC, since they are interested in obtaining UC-security against continuous leakage, they use a more complex variant of the LRCC. We prefer to use the DF-LRCC directly, since it suffices for our needs, and gives a much simpler construction.

– $\mathsf{E}^{\mathrm{DF}} = \left(\mathsf{Enc}^{\mathrm{DF}}, \mathsf{Dec}^{\mathrm{DF}}\right)$, *where:*
  - *For every* $x \in \mathbb{F}^n$, $\mathsf{Enc}^{\mathrm{DF}}\left(x, 1^t, 1^{t_{\mathsf{In}}}\right) = \left(x, \left(\mathsf{Enc}^{\mathsf{In}}_{\mathrm{DF}}\left(0, 1^t\right)\right)^{2t_{\mathsf{In}}}\right)$, *where* $\left(\mathsf{Enc}^{\mathsf{In}}_{\mathrm{DF}}\left(0, 1^t\right)\right)^k$ *denotes* k *random and independent evaluations of* $\mathsf{Enc}^{\mathsf{In}}_{\mathrm{DF}}\left(0, 1^t\right)$.
  - $\mathsf{Dec}^{\mathrm{DF}}\left((x, \boldsymbol{m}), 1^t, 1^{t_{\mathsf{In}}}\right) = x$.
– $\mathsf{Comp}^{\mathrm{DF}}$, *on input an arithmetic circuit* $C : \mathbb{F}^{n_L} \times \mathbb{F}^{n_R} \to \mathbb{F}^k$, *outputs the circuit* $C^{\mathrm{DF}} : \mathbb{F}^{2n_R + n_L + \mathsf{S}(t, n_L, n_R, |C|)} \to \mathbb{F}^k$ *constructed as follows:*
  - *Construct a circuit* $C_1 : \mathbb{F}^{n_R} \times \mathbb{F}^{n_R} \to \mathbb{F}^{n_R}$ *that evaluates the function* $f_1(x, y) = x + y$. *Denote* $s_1 = |C_1|$, *and let* C'_1 *be the circuit obtained from* C_1 *by the transformation of Definition 11. (Notice that if* y *is uniformly random then* C'_1 *outputs a one-time pad encryption of* x.*)*
  - *Construct the circuit* $C_2 : \mathbb{F}^{n_L + n_R} \times \mathbb{F}^{n_R} \to \mathbb{F}^k$ *such that* $C_2\left((z, c), y\right) = C\left(c + y, z\right)$. *Denote* $s_2 = |C_2|$, *and let* C'_2 *be the circuit obtained from* C_2 *by the transformation of Definition 11. (Notice that if* c *is a one-time pad encryption of some value* x *with pad* y, *then* C'_2 *emulates* C *on* x *and* z.*)*
  - *Let* $r = r(t)$ *denote the total length of masking inputs used by a gadget of Construction 6. Then* $\mathsf{S} = \mathsf{S}(t, n_L, n_R, |C|) = r(t) \cdot (s_1 + s_2 + n_L + 4n_R)$. *(Notice that* S *is the number of masking inputs used in* C'_1 *and* C'_2.*)*
  - $C^{\mathrm{DF}}(x, y, z) = C'_2\left(z, \left(C'_1(x, y)\right), y\right)$. *(Intuitively,* C^{DF} *first uses* C'_1 *to encrypt* x *with pad* y, *and then evaluates* C'_2 *on the encryption output by* C'_1, z *and pad* y.*)*

We note that the correctness error of the LTCC of Theorem 3.2 might be abused by malicious parties (e.g., a malicious ZK prover in Sect. 4.1, or malicious parties in Sect. 6) to violate the correctness of the computation, which we overcome by checking whether a correctness error occurred, as described in the following remark.

Remark 1 (Dealing with gadget failures). We will actually use a modified version of Construction 3, in which C^{DF} also computes an error flag, indicating whether the computation failed in one of its gadget (i.e., failed in *all* t *copies of the gadget*, see Remark 3). More specifically, each of the two parties implementing the gadget computes *in the clear* a flag indicating whether its encoding of the output is a valid encoding (i.e., all entries are non-zero), and each party locally combines the flags it generated for all the gadgets. This additional computation is needed since malicious parties (e.g., a malicious prover in the leakage-secure ZK circuit of Construction 4) may *not* choose the masking inputs at random, and might generate them in a "smart" way which will *always* cause gadgets to fail.

We note that thought these flags are generated in the clear, they do not violate the leakage-tolerance property of Construction 3. The reason is these flags are generated locally (by each of the parties), and so could be generated by the leakage function from the simulated wire values which the LT simulator (of Definition 10) generates. This observation gives a reduction from any t-BCL

function on the modified circuit to a t-BCL function on the *original* circuit, and so when using Construction 3 as a building block, we will implicitly disregard these additional wires (remembering that any leakage on the modified circuit *with* the flags can be generated by related leakage on the original circuit). Finally, we note that in an honest execution the flag is set only with negligible probability (and so the fact that the flag is computed in the clear does not violate leakage-resilience).

Remark 2. To combine Construction 3 with Construction 2, we assume that Construction 3 is implemented using a boolean circuit (implementing group operations via operations over $\mathbb{F}_2$) that operates over a standard basis.

Dziembowski and Faust (Corollary 2 in the full version of [15]) show that the DF-LRCC resists OCL leakage, which by the result of [8] implies the existence of an LTCC against such leakage. Combined with Lemma 2 below (which shows a relation between OCL and BCL), we have the following:

Theorem 8 ([15] and [8], and Lemma 2). *Let $t \in \mathbb{N}$ be a leakage bound, and $n, k \in \mathbb{N}$ be input and output length parameters. Then for every polynomial $p(t)$ there exist a finite field $\mathbb{F}$ of size $\Omega(t)$, and a negligible function $\epsilon(t) = \mathsf{negl}(t)$ for which there exists an $\left(\mathcal{L}^{\widetilde{t}}_{\mathrm{BCL}}, \epsilon(t), p(t)\right)$-LTCC, where $\widetilde{t} = 0.16t \log_2 |\mathbb{F}| - 1 - \log_2 |\mathbb{F}|$, and $\mathcal{L}^{T}_{\mathrm{BCL}}$ is the family of all $\widetilde{t}$-BCL functions.*

Theorem 8 relies on the next lemma (whose proof appears in Appendix A) which states that security against so-called "only computation leaks" (OCL) implies security against BCL. (One can also show that $2t$-BCL implies resilience against t-OCL.) Recall that in the context of OCL, the wires of the leakage-resilient circuit $\widehat{C}$ are divided according to some partition $\mathcal{P}$, into two "parts" $\widehat{C}_L, \widehat{C}_R$. The input encodings of $\widehat{C}$ are also divided into two parts, e.g., an encoding $\widehat{x}$ is divided into $\widehat{x}_L$ (which is the input of $\widehat{C}_L$) and $\widehat{x}_R$ (which constitutes the input to $\widehat{C}_R$) The adversary can (adaptively) pick functions $f^L_1, \cdots, f^L_{n_L}$, and $f^R_1, \cdots, f^R_{n_R}$ for some $n_L, n_R \in \mathbb{N}$, where the combined output lengths of $f^L_1, \cdots, f^L_{n_L}$ (and $f^R_1, \cdots, f^R_{n_R}$) is at most t. In the execution of $\widehat{C}$ on $\widehat{x}$, the adversary is given $f^L_i\left[\widehat{C}_L, \widehat{x}_L\right], 1 \leq i \leq n_L$ and $f^R_i\left[\widehat{C}_R, \widehat{x}_R\right], 1 \leq i \leq n_R$, and chooses the next leakage functions based on previous leakage. The output of the leakage is taken to be the combined outputs of all leakage functions $f^L_1, \cdots, f^L_{n_L}, f^R_1, \cdots, f^R_{n_R}$. We say that a circuit is $(\mathcal{L}^t_{\mathrm{OCL}}, \epsilon)$-leakage-resilient with relation to the partition $\mathcal{P} = \left(\widehat{C}_L, \widehat{C}_R\right)$, if the real-world output of the OCL functions can be efficiently simulated (given only the description of the circuit, and its outputs if $\widehat{C}$ computes the outputs in the clear), and the statistical distance between the actual and simulated wire values is at most ϵ. (We refer the reader to, e.g., [15] for a more formal definition of OCL.) We note that we allow the adversary to leak on the two components of the computation in an arbitrary order, a notion which is sometimes referred to as "OCL+".

Lemma 2 (OCL+-resilience implies BCL-resilience). *Let $\epsilon \in (0,1)$ be an error bound, $t \in \mathbb{N}$ be a leakage bound, and C be a boolean circuit. If C is $(\mathcal{L}^t_{\mathrm{OCL}}, \epsilon)$-leakage-resilient with relation to partition $\mathcal{P}$, then C is also $(\mathcal{L}, \epsilon)$-leakage-resilient for the family $\mathcal{L}$ of all t-BCL functions with relation to the same partition $\mathcal{P}$.*

The following property of Construction 3 will be used to guarantee correctness of our constructions in the presence of malicious parties (see Appendix A for the proof).

Lemma 3 (Ill-formed masking inputs correspond to additive attacks). *Let $\mathsf{S} : \mathbb{N}^4 \to \mathbb{N}$ be the length function from Definition 11. Then Construction 3 has the following property. For every circuit $C : \mathbb{F}^{n_L} \times \mathbb{F}^{n_R} \to \mathbb{F}^k$, every security parameter $t \in \mathbb{N}$, and every $\mathsf{m} \in \mathbb{F}^{\mathsf{S}(t,n_L,n_R,|C|)}$, there exists an additive attack $\mathcal{A}_{\mathsf{m}}$ on C such that for every $x \in \mathbb{F}^{n_L+n_R}$, and every $\hat{x} = (x, \mathsf{m})$ it holds that $C^{\mathrm{DF}}(\hat{x}) = C^{\mathcal{A}_{\mathsf{m}}}(x)$. Moreover, there exists a PPT algorithm Alg such that $\mathsf{Alg}(\mathsf{m}) = \mathcal{A}_{\mathsf{m}}$.*

4 Leakage-Secure Zero-Knowledge

In this section we describe our leakage-secure zero-knowledge circuits. At a high level, an $\mathcal{L}$-secure ZK circuit for a family $\mathcal{L}$ of functions is a randomized algorithm Gen that given an error parameter ϵ, and an input length n, outputs a randomized *prover input encoder* Enc_P, and a circuit T. T takes an input from a *prover* P, and returns output to a *verifier* V, and is used by P to convince V that $x \in L_{\mathcal{R}}$. T guarantees soundness, and zero-knowledge even when V obtains leakage from $\mathcal{L}$ on the internals of T.

Definition 12 ($\mathcal{L}$-secure ZK circuit). *Let $\mathcal{R} = \mathcal{R}(x,w)$ be an NP-relation, $\mathcal{L}$ be a family of functions, and $\epsilon > 0$ be an error parameter. We say that* Gen *is an* $\mathcal{L}$-secure zero-knowledge (ZK) circuit *if the following holds.*

- ***Syntax.*** Gen *is a deterministic algorithm that has input $\epsilon, 1^n$, runs in time $\mathsf{poly}(n, \log(1/\epsilon))$, and outputs (Enc_P, T) defined as follows. Enc_P is a randomized circuit that on input (x,w) such that $|x| = n$ (x is the* input*, and w is the* witness*) outputs the* prover input *y for T; and T is a randomized circuit that takes input y and returns $z \in \{0,1\}^{n+1}$.*
- ***Correctness.*** *For every $\epsilon > 0$, every $n \in \mathbb{N}$, and every $(x,w) \in \mathcal{R}$ such that $|x| = n$, $\Pr[T(\mathsf{Enc}_P(x,w)) = (x,1)] \geq 1 - \epsilon$, where $(\mathsf{Enc}_P, T) \leftarrow \mathsf{Gen}(\epsilon, 1^n)$, and the probability is over the randomness used by Enc_P, T.*
- ***Soundness.*** *For every (possibly malicious, possibly unbounded) prover P^*, every $\epsilon > 0$, every $n \in \mathbb{N}$, and any $x \notin L_{\mathcal{R}}$ such that $|x| = n$, $\Pr[T(P^*(x)) = (x,1)] \leq \epsilon$, where $(\mathsf{Enc}_P, T) \leftarrow \mathsf{Gen}(\epsilon, 1^n)$, and the probability is over the randomness used by P^*, T.*
- ***$\mathcal{L}$-Zero-knowledge.*** *For $(x,w) \in \mathcal{R}$ we define the following experiments.*

- *For a (possibly malicious, possibly unbounded) verifier V^*, define the experiment* $\mathsf{Real}_{V^*,\mathsf{Gen}}(x,w,\epsilon)$ *where V^* has input x,ϵ, and chooses a leakage function $\ell \in \mathcal{L}$, and* $\mathsf{Real}_{V^*,\mathsf{Gen}}(x,w,\epsilon) = (T(\mathsf{Enc}_P(x,w)), \ell[T, \mathsf{Enc}_P(x,w)])$, *where* $(\mathsf{Enc}_P, T) \leftarrow \mathsf{Gen}(\epsilon, 1^n)$, *and $[T,y]$ denotes the wires of T when evaluated on y.*
- *For a* simulator *algorithm* Sim *that has input x,ϵ, and* one-time *oracle access to ℓ, the experiment* $\mathsf{Ideal}_{\mathsf{Sim},\mathcal{R}}(x,w,\epsilon)$ *is defined as follows:* $\mathsf{Ideal}_{\mathsf{Sim},\mathcal{R}}(x,w,\epsilon) = \mathsf{Sim}^{\ell}(\epsilon,x)$, *where* $\mathsf{Sim}^{\ell}(\epsilon,x)$ *is the output of* Sim, *given one-time oracle access to ℓ.*

We say that Gen *has $\mathcal{L}$-zero-knowledge ($\mathcal{L}$-ZK) if for every (possibly malicious, possibly unbounded) verifier V^* there exists a simulator* Sim *such that for every $\epsilon > 0$, every $n \in \mathbb{N}$, and every $(x,w) \in \mathcal{R}$ such that $|x| = n$,* $\mathsf{SD}(\mathsf{Real}_{V^*,\mathsf{Gen}}(x,w,\epsilon), \mathsf{Ideal}_{\mathsf{Sim},\mathcal{R}}(x,w,\epsilon)) \leq \epsilon$.

4.1 The Leakage-Secure ZK Circuit

We now construct the leakage-secure ZK circuit by combining the LRCC $\left(\mathsf{Comp}^{\mathrm{GIMSS}}, \mathsf{E}^{\mathrm{GIMSS}}_{\mathsf{Inp}}, \mathsf{Dec}^{\mathrm{GIMSS}}_{\mathsf{Out}}\right)$ of Theorem 7 with the LTCC $\left(\mathsf{Comp}^{\mathrm{DF}}, \mathsf{E}^{\mathrm{DF}}\right)$ of Theorem 8.

At a high level, we compile the verification circuit $C_{\mathcal{R}}$ of an NP-relation $\mathcal{R}$ using $\mathsf{Comp}^{\mathrm{GIMSS}}$, where the prover provides the encoded input and witness for the compiled circuit $\hat{C}_{\mathcal{R}}$. $\hat{C}_{\mathcal{R}}$ has encoded outputs, and only guarantees that BCL leakage cannot distinguish between the executions on two different witnesses. To achieve full-fledged ZK, we use $\mathsf{Comp}^{\mathrm{DF}}$ to decode the outputs of $\hat{C}_{\mathcal{R}}$. Recall that circuits compiled with $\mathsf{Comp}^{\mathrm{DF}}$ have masking inputs, and moreover, their leakage-tolerance property crucially relies on the fact that the masks are *unknown to the leakage function*. Therefore, these masking inputs must be provided by the prover as part of the input encoding (which is generated using Enc_P). However, since the correctness of the computation is guaranteed *only when the masking inputs are well-formed*, a malicious prover P^* can violate soundness by providing ill-formed masking inputs (which were *not* drawn according to the "right" distribution), and thus modify the computed functionality, and potentially cause the circuit to accept $x \notin L_{\mathcal{R}}$. As discussed in Sect. 3.2, the effect of ill-formed masking inputs corresponds to applying an additive attack on the original decoding circuit, so we can protect against such attacks by first replacing the decoding circuit with an AMD circuit.

Construction 4 (Leakage-secure ZK circuit). *Let $n \in \mathbb{N}$ be an input length parameter, $t \in \mathbb{N}$ be a security parameter, and $c \in \mathbb{N}$ be a constant. Let $\mathcal{R} = \mathcal{R}(x,w)$ be an NP-relation, with verification circuit $C_{\mathcal{R}}$ of size $s = |C_{\mathcal{R}}|$. The leakage-secure ZK circuit uses the following building blocks (where any field operations are implemented via bit operations).*

- *The LRCC* $\left(\mathsf{Comp}^{\mathrm{GIMSS}}, \mathsf{E}^{\mathrm{GIMSS}}_{\mathsf{In}} = \left(\mathsf{Enc}^{\mathrm{GIMSS}}_{\mathsf{In}}, \mathsf{Dec}^{\mathrm{GIMSS}}_{\mathsf{In}}\right), \mathsf{Dec}^{\mathrm{GIMSS}}_{\mathsf{Out}}\right)$ *of Theorem 7 (Construction 2), and its underlying small-bias encoding scheme* $\left(\mathsf{Enc}^{\oplus} : \mathbb{F}_2 \times \mathbb{F}_2^c \to \mathbb{F}_2^{2c}, \mathsf{Dec}^{\oplus} : \mathbb{F}_2^{2c} \to \mathbb{F}_2^2\right)$.

- *The LTCC* $\left(\mathsf{Comp}^{\mathrm{DF}}, \mathsf{E}^{\mathrm{DF}}\right)$ *of Theorem 8 (Construction 3) over a field* $\mathbb{F} = \Omega(t)$*, and its underlying encoding scheme* $\mathsf{E}^{\mathsf{In}}_{\mathrm{DF}} = \left(\mathsf{Enc}^{\mathsf{In}}_{\mathrm{DF}}, \mathsf{Dec}^{\mathsf{In}}_{\mathrm{DF}}\right)$.
- *The additively-secure circuit compiler* $\mathsf{Comp}^{\mathsf{add}}$ *of Theorem 4.*
- *The AMD encoding scheme* $\left(\mathsf{Enc}^{\mathsf{amd}}, \mathsf{Dec}^{\mathsf{amd}}\right)$ *of Theorem 5, with encodings of length* $\hat{n}^{\mathsf{amd}}(n,t)$.

On input $1^n, 1^t$, Gen *outputs* (Enc_P, T) *defined as follows.*

- *For every input* $x \in \{0,1\}^n$*, and witness* w, $\mathsf{Enc}_P(x,w) = \left(\mathsf{Enc}_{\mathrm{GIMSS}}\left((x,w),1^t\right), \mathsf{Enc}^{\mathsf{In}}_{\mathrm{DF}}\left(0^{s'}, 1^t\right)\right)$ *for a parameter* s' *whose value is set below.*
- *Let* n_w *be a bound on the maximal witness length for inputs of length* n*.* T *is obtained by concatenating the* decoding component T'' *to the* verification component C'' *(namely, applying* T'' *to the outputs of* C''*) which are defined next.*
  1. ***The verification component*** C''**.** *Define* $C' : \mathbb{F}_2^{n+n_w} \to \mathbb{F}_2^{n+1}$ *as* $C'(x,w) = (x, C_{\mathcal{R}}(x,w))$*. Let* C'_2 *denote the circuit that emulates* C'*, but replaces each output bit with (the bit string representation of) the bit as an element of* $\mathbb{F}$*. Then* $C'' = \mathsf{Comp}^{\mathrm{GIMSS}}(C'_2)$.
  2. ***The decoding component.***
     - *Construct the circuit* $C^{\mathsf{amd}} : \mathbb{F}^{2c\cdot t\cdot(n+1)} \to \mathbb{F}^{\hat{n}^{\mathsf{amd}}(n+1,t)}$ *that operates as follows:*
       - *Decodes its input using* $\mathsf{Dec}^{\mathrm{GIMSS}}_{\mathsf{Out}}$ *to obtain the output* (f,x,z)*.*
       - *If* $f = 1$*,* $x \notin \{0,1\}^n$*, or* $z \neq 1$*, then* C^{amd} *sets* $z' = 0$*. Otherwise, it sets* $z' = 1$*.*
       - *Generates* $\boldsymbol{e} \leftarrow \mathsf{Enc}^{\mathsf{amd}}\left((x,z'),1^t\right)$*, and outputs* $\boldsymbol{e}$*.*
     - *Generate* $\widehat{C}^{\mathsf{amd}} = \mathsf{Comp}^{\mathsf{add}}\left(C^{\mathsf{amd}}\right)$.
     - *Generate* $T' = \mathsf{Comp}^{\mathrm{DF}}\left(\widehat{C}^{\mathsf{amd}}\right)$*. Let* s' *denote the number of masking inputs used in* T'*.*
     - *Construct the circuit* T'' *that on input* y*, operates as follows:*
       - *Computes* $(f_L, f_R, \boldsymbol{e}) = T'(y)$*. (Recall that* f_L, f_R *are flags indicating whether a gadget of* T' *has failed.)*
       - *Computes* $(f,x,z) = \mathsf{Dec}^{\mathsf{amd}}(\boldsymbol{e}, 1^t)$*, where* $f,z \in \mathbb{F}$ *and* $x \in \mathbb{F}^n$*. If* $f = f_L = f_R = 0$*,* $x \in \{0,1\}^n$*, and* $z = 1$ *then* T' *outputs* $(x,1)$*. Otherwise, it outputs* 0^{n+1}.

We show in the full version [20] that Construction 4 is a leakage-secure ZK circuit, proving Theorems 1 and 2 (for Theorem 1, we have the prover provide the masking inputs used for the computation in C'', while the verifier provides the randomness used in T'').

5 Multiparty LRCCs: Definition

In this section we define the notion of multiparty LRCCs, a generalization of leakage-secure ZK circuits to evaluation of general functions with $m \geq 1$ parties. We first formalize the notion of secure computation with a single piece of trusted

(but leaky) hardware device, where security with abort holds in the presence of adversaries that corrupt a subset of parties, and obtain leakage (from a predefined leakage class) on the internals of the device. This raises the following points.

1. The output should include a flag signaling whether there was an abort.
2. Leakage on the wires of the device should reveal nothing about the internal computations, or the inputs of the honest parties, other than what can be computed from the output. This necessitates randomized computation.
3. The inputs should be encoded, otherwise leakage on the input wires may reveal information that cannot be computed from the outputs. This should be contrasted with the ZK setting, in which x is assumed to be public, and so when all parties are honest the output is $(x, 1)$ and can therefore be computed in the clear.

To guarantee that an adversary that only obtains leakage on the internals of the device (but does not corrupt any parties) learns nothing about the inputs or internal computations, the outputs must be encoded. Therefore, the device, which is implemented as a circuit, is associated with an input encoding algorithm Enc, and an output decoding algorithm Dec. The above discussion is formalized in the next definition.

Definition 13 (Secure function implementation). *Let $m \in \mathbb{N}$, $f : (\{0,1\}^n)^m \to \{0,1\}^k$ be an m-argument function, $\mathcal{L}$ be a family of leakage functions, and $\epsilon > 0$. We say that $(\mathsf{Enc}, C, \mathsf{Dec})$ is an m-party $(\mathcal{L}, \epsilon)$-secure implementation of f if it satisfies the following requirements.*

- ***Syntax:***
  - $\mathsf{Enc} : \{0,1\}^n \to \{0,1\}^{\hat{n}}$ *is a randomized function, called* the input encoder.
  - $C : \left(\{0,1\}^{\hat{n}}\right)^m \to \{0,1\}^{\hat{k}}$ *is a randomized circuit.*
  - $\mathsf{Dec} : \{0,1\}^{\hat{k}} \to \{0,1\}^{k+1}$ *is a deterministic function called* the output decoder.
- ***Correctness.*** *For every $x_1, \cdots, x_m \in \{0,1\}^n$,*

$$\Pr\left[\mathsf{Dec}\left(C\left(\mathsf{Enc}\left(x_1\right), \cdots, \mathsf{Enc}\left(x_m\right)\right)\right) = \left(0, f\left(x_1, \cdots, x_m\right)\right)\right] \geq 1 - \epsilon.$$

- ***Security.*** *For every adversary $\mathcal{A}$ there exists a simulator Sim such that for every input $(x_1, \cdots, x_m) \in (\{0,1\}^n)^m$, and every leakage function $\ell \in \mathcal{L}$, $\mathsf{SD}(\mathsf{Real}, \mathsf{Ideal}) \leq \epsilon$, where $\mathsf{Real}, \mathsf{Ideal}$ are defined as follows.*
  Real*:*
  - *$\mathcal{A}$ picks a set $\mathsf{B} \subset [m]$ of corrupted parties, and (possibly ill-formed) encoded inputs $x'_i \in \{0,1\}^{\hat{n}}$ for every $i \in \mathsf{B}$.*
  - *For every uncorrupted party $j \notin \mathsf{B}$, let $x'_j = \mathsf{Enc}\left(x_j\right)$.*
  - *If $\mathsf{B} \neq \emptyset$ then $z = \left(C\left(x'_1, \cdots, x'_m\right), \mathsf{Dec}\left(C\left(x'_1, \cdots, x'_m\right)\right)\right)$, otherwise z is empty. (Intuitively, z represents the information $\mathcal{A}$ has about the output of C. If $\mathsf{B} = \emptyset$ then $\mathcal{A}$ learns nothing.)*
  - *$\mathsf{Real} = \left(\mathsf{B}, \{x'_i\}_{i \in \mathsf{B}}, \ell\left[C, \left(x'_1, \cdots, x'_m\right)\right], z\right)$.*

Ideal*:*

- Sim *picks a set* $\mathsf{B} \subset [m]$ *of corrupted parties and receives their inputs* $\{x_i\}_{i\in\mathsf{B}}$. Sim *then chooses effective inputs* $w_i \in \{0,1\}^n$ *for every* $i \in \mathsf{B}$, *and if* $\mathsf{B} \neq \emptyset$ *obtains* $f(w_1 \cdots, w_m)$, *where* $w_j = x_j$ *for every* $j \notin \mathsf{B}$.
- Sim *chooses* $b \in \{0,1\}$. *(Intuitively,* b *indicates whether to abort the computation.)*
- *If* $\mathsf{B} \neq \emptyset$ *and* $b = 0$, *set* $y = (0, f(w_1, \cdots, w_m))$, *if* $\mathsf{B} \neq \emptyset$ *and* $b = 1$, *set* $y = \left(1, 0^k\right)$, *and if* $\mathsf{B} = \emptyset$ *then* y *is empty.*
- *Let* $\left(W, \{x'_i\}_{i\in\mathsf{B}}\right)$ *denote the output of* Sim, *where* W *contains a bit for each wire of* C, *and* $x'_i \in \{0,1\}^{\hat{n}}$ *for every* $i \in \mathsf{B}$. *Denote the restriction of* W *to the output wires by* W_{Out}.
- *If* $\mathsf{B} \neq \emptyset$, *let* $z = (W_{\mathsf{Out}}, y)$. *Otherwise,* z *is empty.*
- $\mathsf{Ideal} = \left(\mathsf{B}, \{x'_i\}_{i\in\mathsf{B}}, \ell(W), z\right)$.

We say that $(\mathsf{Enc}, C, \mathsf{Dec})$ *is a* passive-secure implementation of f *if the security property holds with the following modifications: (1)* $\mathcal{A}$ *does not choose* $x'_i, i \in \mathsf{B}$, *and instead,* $x'_i \leftarrow \mathsf{Enc}(x_i)$ *for every* $i \in \mathsf{B}$*; and (2)* Sim *always chooses* $b = 0$.

We now define an m-party LRCC which, informally, is an asymptotic version of Definition 13.

Definition 14 (m-party circuit). *Let* $m \in \mathbb{N}$. *We say that a boolean circuit* C *is an* m-party circuit *if its input can be partitioned into* m *equal-length strings, i.e.,* $C : (\{0,1\}^n)^m \to \{0,1\}^k$ *for some* $n, k \in \mathbb{N}$.

Definition 15 (Multiparty LRCCs and passive-secure multiparty LRCCs). *Let* $m \in \mathbb{N}$, $\mathcal{L}$ *be a family of leakage functions,* $\mathsf{S}(n)$ *be a size function, and* $\epsilon(n) : \mathbb{N} \to \mathbb{R}^+$. *Let* Comp *be a PPT algorithm that on input* m, *and an* m*-party circuit* $C : (\{0,1\}^n)^m \to \{0,1\}^k$, *outputs a circuit* $\hat{C}$.

We say that $(\mathsf{Enc}, \mathsf{Comp}, \mathsf{Dec})$ *is an* m*-party* $(\mathcal{L}, \epsilon(n), \mathsf{S}(n))$-leakage-resilient circuit compiler *(*m-party LRCC, or multiparty LRCC*) if there exists a PPT simulator* Sim *such that for all sufficiently large* n*'s, and every* m*-party circuit* $C : (\{0,1\}^n)^m \to \{0,1\}^k$ *of size at most* $\mathsf{S}(n)$ *that computes a function* f_C, $\left(\mathsf{Enc}, \hat{C}, \mathsf{Dec}\right)$ *is an* $(\mathcal{L}, \epsilon(n))$*-secure implementation of* f_C, *where the security property holds with simulator* Sim *that is given the description of* C, *and has black-box access to the adversary. We say that* $(\mathsf{Enc}, \mathsf{Comp}, \mathsf{Dec})$ *is a* passively-secure m-party $(\mathcal{L}, \epsilon(n), \mathsf{S}(n))$-LRCC *if* $\left(\mathsf{Enc}, \hat{C}, \mathsf{Dec}\right)$ *is an* $(\mathcal{L}, \epsilon(n))$*-passively-secure implementation of* f_C, *where security holds with simulator* Sim.

Remark 1. *Definitions 13–15 naturally extend to the arithmetic setting in which* C *is an arithmetic circuit over a finite field* $\mathbb{F}$. *When discussing the arithmetic setting, we explicitly state the field over which we are working (e.g., we use "multiparty LRCC over* $\mathbb{F}$*" to denote that the multiparty LRCC is in the arithmetic setting with field* $\mathbb{F}$*).*

6 A Multiparty LRCC

In this section we construct a multiparty LRCC that withstands active adversaries. The high-level idea of the construction is as follows. Given an m-party protocol C, we first replace it with a circuit C^{share} that emulates C but outputs a secret-sharing of the outputs, then compile C^{share} using the LRCC of [25]. We then refresh each of the shares using a circuit C_{Dec}. However, to guarantee leakage-resilience, and correctness of the computation in the presence of actively-corrupted parties, we first replace the circuit C_{Dec} with its additively-secure version C'_{Dec}, then compile C'_{Dec} using the LTCC of [15] to obtain a leakage-tolerant circuit $\widehat{C}'_{\mathsf{Dec}}$. We use m copies of $\widehat{C}'_{\mathsf{Dec}}$, where the i'th copy refreshes the i'th secret share, using masking inputs provided by the i'th party. Each party provides, as its input encoding to the device, both a leakage-resilient encoding of its input, and the masking inputs needed for the computation in $\widehat{C}_{\mathsf{Dec}}$. The output decoder decodes each of the secret shares, and reconstructs the output from the shares (unless it detects that one of the parties provided ill-formed masking inputs, in which case the computation aborts). This is formalized in the next construction.

Construction 5 (Multiparty LRCC). *Let $m \in \mathbb{N}$ denote the number of parties, $t \in \mathbb{N}$ be a security parameter, $n \in \mathbb{N}$ be an input length parameter, $k \in \mathbb{N}$ be an output length parameter, and $c \in \mathbb{N}$ be a constant. The m-party LRCC uses the following building blocks:*

- *The LRCC $\left(\mathsf{Comp}^{\mathrm{GIMSS}}, \mathsf{E}^{\mathrm{GIMSS}}_{\mathsf{In}} = \left(\mathsf{Enc}^{\mathrm{GIMSS}}_{\mathsf{In}}, \mathsf{Dec}^{\mathrm{GIMSS}}_{\mathsf{In}}\right), \mathsf{Dec}^{\mathrm{GIMSS}}_{\mathsf{Out}}\right)$ of Theorem 7 (Construction 2), where the outputs of the leakage-resilient circuit are encoded by the encoding scheme $\left(\mathsf{Enc}_{\mathrm{GIMSS}} : \mathbb{F}_2 \to \mathbb{F}_2^{4ct}, \mathsf{Dec}_{\mathrm{GIMSS}} : \mathbb{F}_2^{4ct} \to \mathbb{F}_2^2\right)$.*
- *The LTCC $\left(\mathsf{Comp}^{\mathrm{DF}}, \mathsf{E}^{\mathrm{DF}}\right)$ of Theorem 8 (Construction 3) over a field $\mathbb{F} = \Omega(t)$, and its underlying encoding scheme $\mathsf{E}^{\mathsf{In}}_{\mathrm{DF}} = \left(\mathsf{Enc}^{\mathsf{In}}_{\mathrm{DF}}, \mathsf{Dec}^{\mathsf{In}}_{\mathrm{DF}}\right)$ that outputs encodings of length $\hat{n}^{\mathrm{DF}}(n,t)$.*
- *The additively-secure circuit compiler $\mathsf{Comp}^{\mathsf{add}}$ of Theorem 4.*

The m-party LRCC $(\mathsf{Enc}, \mathsf{Comp}, \mathsf{Dec})$ *is defined as follows.*

- *For every $n, t, t_{\mathsf{In}} \in \mathbb{N}$ and every $x \in \mathbb{F}^n$, $\mathsf{Enc}(x, 1^t, 1^{t_{\mathsf{In}}}) = \left(\mathsf{Enc}^{\mathrm{GIMSS}}_{\mathsf{In}}(x, 1^t, 1^{t_{\mathsf{In}}}), \mathsf{Enc}^{\mathrm{DF}}_{\mathsf{In}}(0^{t_{\mathsf{In}}}, 1^t)\right)$.*
- *For every $y = \left(\left(f^1_L, f^1_R, y^1\right), \cdots, \left(f^m_L, f^m_R, y^m\right)\right) \in \left(F^{2+2tc(k+1)}\right)^m$, $\mathsf{Dec}(y, 1^t)$ computes $\left(f_i, z^i\right) = \mathsf{Dec}^{\mathsf{Out}}_{\mathrm{GIMSS}}\left(y^i, 1^t\right)$. If $f^i_L = f^i_R = f_i = 0$ for all $1 \le i \le m$ then Dec outputs $\left(0, \sum_{i=1}^m z^i\right)$, otherwise it outputs $\left(1, 0^k\right)$. (Intuitively, each triplet $\left(f^i_L, f^i_R, y^i\right)$ consists of a pair of flags output by the LTCC, indicating whether the computation in one of its gadgets failed; and an encoding of a flag, concatenated with an additive secret share of the output.)*
- *Comp on input $m \in \mathbb{N}$, and an m-party circuit $C : (\mathbb{F}^n)^m \to \mathbb{F}^k$:*
  1. *Constructs the circuit $C^{\mathsf{share}} : (\mathbb{F}^n)^m \to \mathbb{F}^{mk}$ that operates as follows:*

- *Evaluates* C *on inputs* $x_1, \cdots, x_m$ *to obtain the output* $y = C(x_1, \cdots, x_m)$.
- *Generates* $y_1, \cdots, y_{m-1} \in_R \mathbb{F}^k$, *and sets* $y_m = y \oplus \sum_{i=1}^{m-1} y_i$. *(*$y_1, \cdots, y_m$ *are random additive secret shares of* y.*)*
- *For every* $1 \leq i \leq m$, *generates* y'_i *by replacing each bit of* y_i *with (the bit string representation of) the bit as an element of* $\mathbb{F}$.
- *Outputs* $(y'_1, \cdots, y'_m)$.

2. *Computes* $C' = \mathsf{Comp}^{\mathrm{GIMSS}}\left(C^{\mathsf{share}}\right)$.
3. *Construct the circuit* $C^{\mathsf{Dec}} : \mathbb{F}^{4ct(k+1)} \to \mathbb{F}^{4ct(k+1)}$ *that operates as follows:*
   - *Decodes its input using* $\mathsf{Dec}_{\mathsf{Out}}^{\mathrm{GIMSS}}$ *to obtain a flag* $f \in \mathbb{F}_2$ *and output* $z \in \mathbb{F}^k$.
   - *If* $f = 1$, *sets* $z' = 0^k$, *otherwise* $z' = z$.
   - *Generates* $\boldsymbol{e} \leftarrow \mathsf{Enc}_{\mathrm{GIMSS}}\left((f, z'), 1^t\right)$, *and outputs* $\boldsymbol{e}$.
4. *Generate* $\widehat{C}^{\mathsf{amd}} = \mathsf{Comp}^{\mathsf{add}}\left(C^{\mathsf{Dec}}\right)$.
5. *Generate* $C'' = \mathsf{Comp}^{\mathrm{DF}}\left(\widehat{C}^{\mathsf{amd}}\right)$.
6. *Outputs the circuit* $\hat{C}$ *obtained by concatenating a copy of* C'' *to each of the* m *outputs of* C'. *(We note that the* i*'th copy of* C'' *takes its masking inputs from the encoding of the* i*'th input to* $\hat{C}$.*)*

The next theorem (whose proof appears in the full version [20]) states that Construction 5 is a multiparty LRCC.

Theorem 9 (Multiparty LRCC). *Let* $n, k \in \mathbb{N}$ *be input and output length parameters,* $\boldsymbol{S}(n) : \mathbb{N} \to \mathbb{N}$ *be a size function,* $\epsilon(n), \epsilon'(n) : \mathbb{N} \to (0,1)$ *be error functions,* $t \in \mathbb{N}$ *be a leakage bound, let* $c \in \mathbb{N}$ *be a constant, and let* $\mathsf{m} \in \mathbb{N}$ *denote the number of parties. Let* $\mathcal{L}$ *denote the family of all* t*-BCL functions. If:*

- $\left(\mathsf{Comp}^{\mathrm{GIMSS}}, \mathsf{Enc}_{\mathsf{In}}^{\mathrm{GIMSS}}, \mathsf{Dec}_{\mathsf{Out}}^{\mathrm{GIMSS}}\right)$ *is an* $(\mathcal{L}, \epsilon, \boldsymbol{S}(n) + 2m)$*-relaxed LRCC with abort, where for security parameter* t, $\mathsf{Dec}_{\mathsf{Out}}^{\mathrm{GIMSS}}, \mathsf{Enc}_{\mathrm{GIMSS}}$ *can be evaluated using circuits of size* $s^{\mathrm{GIMSS}}(t)$,
- $\mathsf{Comp}^{\mathsf{add}}$ *is an* $\epsilon'(n)$*-additively-secure circuit compiler over* $\mathbb{F}$, *where there exist: (1)* $B : \mathbb{N} \to \mathbb{N}$ *such that for any circuit* C, $\mathsf{Comp}^{\mathsf{add}}(C)$ *has size at most* $B(|C|)$*; and (2) a PPT algorithm* Alg' *that given an additive attack* $\mathcal{A}$ *outputs the ideal attack* $\left(\mathbf{a}^{\mathsf{in}}, \mathcal{A}^{\mathsf{Out}}\right)$ *(whose existence follows from the additive-attack security property of Definition 3), and*
- $\left(\mathsf{Comp}^{\mathrm{DF}}, \mathsf{E}^{\mathrm{DF}}\right)$ *is an* $\left(\mathcal{L}, \epsilon, B\left(2s^{\mathrm{GIMSS}}(t) + ck\right)\right)$*-LTCC.*

Then Construction 5 is an m*-party* $(\mathcal{L}, (2m+1)\epsilon(n) + \epsilon'(n) + \mathsf{negl}(t), \boldsymbol{S}(n))$*-LRCC.*

Moreover, if on input a circuit of size s, $\mathsf{Comp}^{\mathrm{GIMSS}}, \mathsf{Comp}^{\mathrm{DF}}$ *output circuits of size* $\widehat{s}^{\mathrm{GIMSS}}(s)$, *and* $s^{\mathrm{DF}}(s)$, *respectively, then on input a circuit* C *of size* s, *the compiler of Construction 5 outputs a circuit* $\widehat{C}$ *of size* $\widehat{s}^{\mathrm{GIMSS}}(s + 2m) + s^{\mathrm{DF}}\left(B\left(2s^{\mathrm{GIMSS}}(t) + ck\right)\right)$.

In the full version, we use Theorem 9 to prove Theorem 3. We also provide a (somewhat) more efficient MPCC construction for passive corruptions.

Acknowledgments. This work was supported in part by the 2017–2018 Rothschild Postdoctoral Fellowship; by the Warren Center for Network and Data Sciences; by the financial assistance award 70NANB15H328 from the U.S. Department of Commerce, National Institute of Standards and Technology; and by the Defense Advanced Research Project Agency (DARPA) under Contract #FA8650-16-C-7622. The second author was supported in part by NSF-BSF grant 2015782, BSF grant 2012366, ISF grant 1709/14, ERC grant 742754, DARPA/ARL SAFEWARE award, NSF Frontier Award 1413955, NSF grants 1619348, 1228984, 1136174, and 1065276, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. This material is based upon work supported by the DARPA through the ARL under Contract W911NF-15-C-0205. The views expressed are those of the authors and do not reflect the official policy or position of the DoD, the NSF, or the U.S. Government. This work was supported in part by NSF grants CNS-1314722, CNS-1413964.

A Gadgets for the LRCC of [15]

In this section we describe the gadgets used in the LRCC of [15], and prove Lemmas 2 and 3.

Construction 6 (Gadgets for an LRCC, [15]). *Let $\mathbb{F}$ be a finite field, and $\mathsf{E}_{\mathsf{IP}} = (\mathsf{Enc}_{\mathsf{IP}}, \mathsf{Dec}_{\mathsf{IP}})$ denote the inner product encoding over $\mathbb{F}$ of Definition 9. Each gadget consists of a* left component C^L*, and a* right component C^R *that are connected to each other. We use the term "X is sent from component Y to component Z" to denote that the value X computed in component Y is the input to some sub-computation performed in component Z.*

1. ***Refresh gadget:***[9] *inputs $\left(\boldsymbol{a}^L, \boldsymbol{a}^R\right) \in \mathsf{Enc}_{\mathsf{IP}}(a, 1^{t^2})$ for $a \in \mathbb{F}$, and masking inputs $\left(\left(\boldsymbol{r}^{L,1}, \boldsymbol{r}^{L,2}\right), \left(\boldsymbol{r}^{R,1}, \boldsymbol{r}^{R,2}\right)\right) \in \mathsf{Enc}^{\mathsf{In}}_{\mathrm{DF}}(0, 1^{t^2})$; outputs $\left(\boldsymbol{a}^{L\prime}, \boldsymbol{a}^{R\prime}\right) \in \mathsf{Enc}_{\mathsf{IP}}(a, 1^{t^2})$.*
   (a) *C^L generates $\boldsymbol{b} \in \mathbb{F}^{t^2}$ such that $\boldsymbol{b}_i = \left(\boldsymbol{a}_i^L\right)^{-1} \times \boldsymbol{r}_i^{L,1}$ for every $1 \le i \le t^2$, and sends $\boldsymbol{b}$ to C^R.*
   (b) *C^R computes $\boldsymbol{c} \in \mathbb{F}^{t^2}$ such that $\boldsymbol{c}_i = \boldsymbol{b}_i \times \boldsymbol{r}_i^{R,1}$ for every $1 \le i \le t^2$.*
   (c) *C^R computes $\boldsymbol{a}^{R\prime} = \boldsymbol{a}^R + \boldsymbol{c}$.*
   (d) *C^R generates $\boldsymbol{d} \in \mathbb{F}^{t^2}$ such that $\boldsymbol{d}_i = \left(\boldsymbol{a}_i^{R\prime}\right)^{-1} \times \boldsymbol{r}_i^{R,2}$ for every $1 \le i \le t^2$, and sends $\boldsymbol{d}$ to C^L.*
   (e) *C^L computes $\boldsymbol{e} \in \mathbb{F}^{t^2}$ such that $\boldsymbol{e}_i = \boldsymbol{d}_i \times \boldsymbol{r}_i^{L,2}$ for every $1 \le i \le t^2$.*
   (f) *C^L computes $\boldsymbol{a}^{L\prime} = \boldsymbol{a}^L + \boldsymbol{e}$.*
2. ***Multiplication by constant gadget:*** *inputs constant $c \in \mathbb{F} \setminus \{0\}$, and $\left(\boldsymbol{a}^L, \boldsymbol{a}^R\right) \in \mathsf{Enc}_{\mathsf{IP}}\left(a, 1^t\right)$ for $a \in \mathbb{F}$; output $\left(\boldsymbol{b}^L, \boldsymbol{b}^R\right) \in \mathsf{Enc}_{\mathsf{IP}}\left(c \times a, 1^t\right)$.*
   (a) *C^L computes $\boldsymbol{b}_i^L = c \times \boldsymbol{a}_i^L$ for every $1 \le i \le t$.*
   (b) *C^R sets $\boldsymbol{b}^R = \boldsymbol{a}^R$.*

[9] This refresh gadget is a simpler construction than the original gadget of [15], due to [1].

3. ***Addition by constant gadget:*** *inputs constant* $c \in \mathbb{F}$*, and* $\left(\boldsymbol{a}^L, \boldsymbol{a}^R\right) \in \mathsf{Enc}_{\mathsf{IP}}\left(a, 1^t\right)$ *for* $a \in \mathbb{F}$*; output* $\left(\boldsymbol{b}^L, \boldsymbol{b}^R\right) \in \mathsf{Enc}_{\mathsf{IP}}\left(c+a, 1^t\right)$*.*
   (a) C^L *sets* $\boldsymbol{b}^L = \boldsymbol{a}^L$*, and sends* $\boldsymbol{a}_1^L$ *to* C^R*.*
   (b) C^R *sets* $\boldsymbol{b}^R = \boldsymbol{a}^R + \left(\left(\boldsymbol{a}_1^L\right)^{-1} \times c, 0, \cdots, 0\right)$*.*
4. ***Generalized multiplication gadget:*** *inputs a constant* $c \in \mathbb{F}$*,* $\left(\boldsymbol{a}^L, \boldsymbol{a}^R\right) \in \mathsf{Enc}_{\mathsf{IP}}\left(a, 1^t\right), \left(\boldsymbol{b}^L, \boldsymbol{b}^R\right) \in \mathsf{Enc}_{\mathsf{IP}}\left(b, 1^t\right)$ *for* $a, b \in \mathbb{F}$*, and masking inputs* $\left(\left(\boldsymbol{r}^{L,1}, \boldsymbol{r}^{L,2}\right), \left(\boldsymbol{r}^{R,1}, \boldsymbol{r}^{R,2}\right)\right) \in \mathsf{Enc}_{\mathrm{DF}}^{\mathsf{In}}\left(0, 1^t\right)$*; output* $\left(\boldsymbol{c}^L, \boldsymbol{c}^R\right) \in \mathsf{Enc}_{\mathsf{IP}}\left(c - a \times b, 1^t\right)$*.*
   (a) C^L *generates a* $t \times t$ *Matrix* $\boldsymbol{L} = \boldsymbol{a}^L \left(\boldsymbol{b}^L\right)^T = \left(a_i^L \times b_j^L\right)_{i,j \in [t]}$*. We interpret* L *as a length-*t^2 *vector.*
   (b) C^R *enerates a* $t \times t$ *Matrix* $\boldsymbol{R} = \boldsymbol{a}^R \left(\boldsymbol{b}^R\right)^T = \left(a_i^R \times b_j^R\right)_{i,j \in [t]}$*. We interpret* R *as a length-*t^2 *vector.*
   (c) C^L, C^R *evaluate the Refresh gadget with inputs* $\boldsymbol{L}, \boldsymbol{R}$*, and masking inputs* $\left(\left(\boldsymbol{r}^{L,1}, \boldsymbol{r}^{L,2}\right), \left(\boldsymbol{r}^{R,1}, \boldsymbol{r}^{R,2}\right)\right)$*, to obtain* $\boldsymbol{L}', \boldsymbol{R}'$ *(which are length-*t^2 *vectors).*
   (d) C^L *sends* $L'_1, L'_{t+1}, \cdots L'_{t^2}$ *to* C^R*.*
   (e) C^R *computes* $d = \langle \left(L'_{t+1}, \cdots L'_{t^2}\right), \left(R'_{t+1}, \cdots, R'_{t^2}\right) \rangle$*.*
   (f) C^R *computes* $\boldsymbol{c}^R = -\left(R'_1, \cdots, R'_t\right) + \left(\left(L'_1\right)^{-1}(c-d), 0, \cdots, 0\right)$*.*
   (g) C^L *computes* $\boldsymbol{c}^L = \left(L'_1, \cdots, L'_t\right)$*.*

Remark 3 (Amplifying correctness). The execution in each gadget can fail (if the generated encodings are not valid inner-product encodings). However, [15] show that for $|\mathbb{F}| = \Omega(t)$, if each computation step is implemented using t copies of the corresponding gadget (and the output of the computation step is set to the output of the first gadget whose output is valid), then each computation step succeeds except with $\mathsf{negl}(t)$ probability. In what follows, we implicitly assume that each computation step is implemented using this amplification technique over t gadgets.

We now prove Lemmas 2 and 3.

Proof (of Lemma 2 (sketch)). Let ℓ be a t-BCL function that corresponds to a two party protocol Π, defined in relation to partition $\mathcal{P}$. Let $\mathsf{NextMsg}_L, \mathsf{NextMsg}_R$ be the next-message functions defining the messages the parties send, given their current view, and assume without loss of generality that the left party sends the first message in the protocol. Let $(\widehat{x}_L, \widehat{x}_R)$ be the input on which $\widehat{C}$ is evaluated, and denote $\mathcal{W}_L = \left[\widehat{C}_L, \widehat{x}_L\right]$, and $\mathcal{W}_R = \left[\widehat{C}_R, \widehat{x}_R\right]$.

To generate the transcript of Π, the adversary operates as follows. First, it picks $f_1^L(z) = \mathsf{NextMsg}_L(z)$. Then, given $f_1^L(\mathcal{W}_L)$, which is the first message that the left party sends in Π, it picks f_1^R to be the function which $\mathsf{NextMsg}_R$ computes, conditioned on the event that $f_1^L(\mathcal{W}_L)$ was the first message which the right party received, and sends f_1^R, to be evaluated on $\mathcal{W}_R$. The adversary continues in this way until all messages of Π have been computed. Since Π is

t-bounded, then in particular each of the two participating parties sends at most t bits, namely the leakage functions we have defined leak at most t bits on each of $\mathcal{W}_L, \mathcal{W}_R$. Therefore, the t-OCL resilience of C guarantees that the leakage can be efficiently simulated, up to statistical distance ϵ.

Proof (of Lemma 3). We analyze the effect of ill-formed masking inputs m in the gadgets of Construction 6, and show that they correspond to applying an additive attack on the underlying gate.

- **Refresh gadget.** Denote $m = \langle \mathbf{r}^{L,1}, \mathbf{r}^{R,1} \rangle + \langle \mathbf{r}^{L,2}, \mathbf{r}^{R,2} \rangle$ (which, if the masking inputs are ill-formed, may not be 0). Then the output of the gadget encodes the value $\langle \mathbf{a}^{L\prime}, \mathbf{a}^{R\prime} \rangle$. We analyze this value. $\langle \mathbf{a}^{L\prime}, \mathbf{a}^{R\prime} \rangle = \sum_{i=1}^{t^2} a_i^{L\prime}, a_i^{r\prime}$ which, by the definition of $\mathbf{a}^{L\prime}, \mathbf{a}^{L\prime}$ is equal to

$$\sum_{i=1}^{t^2} \left(a_i^L + e_i\right)\left(a_i^R + c_i\right) = \sum_{i=1}^{t^2} a_i^L a_i^R + \sum_{i=1}^{t^2} e_i \left(a_i^R + c_i\right) + \sum_{i=1}^{t^2} a_i^L c_i$$

$$= a + \sum_{i=1}^{t^2} e_i a_i^{R\prime} + \sum_{i=1}^{t^2} a_i^L c_i$$

  which, by the definition of $\mathbf{c}, \mathbf{e}$, is equal to

$$a + \sum_{i=1}^{t^2} \left(a_i^{R\prime}\right)^{-1} r_i^{R,2} r_i^{L,2} a_i^{R\prime} + \sum_{i=1}^{t^2} a_i^L \left(a_i^{L,1}\right)^{-1} r_i^{L,1} r_i^{R,1} = a + \langle \mathbf{r}^{L,1}, \mathbf{r}^{R,1} \rangle + \langle \mathbf{r}^{L,2}, \mathbf{r}^{R,2} \rangle$$

  which is equal to $a + m$. Moreover, notice that m can be efficiently computed from $\mathbf{r}^{L,1}, \mathbf{r}^{R,1}, \mathbf{r}^{L,2}, \mathbf{r}^{R,2}$ by computing $\langle \mathbf{r}^{L,1}, \mathbf{r}^{R,1} \rangle + \langle \mathbf{r}^{L,2}, \mathbf{r}^{R,2} \rangle$.
- **Generalized multiplication gadget.** Denote $m = \langle \mathbf{r}^{L,1}, \mathbf{r}^{R,1} \rangle + \langle \mathbf{r}^{L,2}, \mathbf{r}^{R,2} \rangle$. The output of the gadget encodes the value $\langle \mathbf{c}^L, \mathbf{c}^R \rangle = \sum_{i=1}^{t} c_i^L c_i^R$ which, by the definition of $\mathbf{c}^L, \mathbf{c}^R$, is equal to

$$L_1' \left(-R_1' + \left(L_1'\right)^{-1} (c - d)\right) + \sum_{i=2}^{t} L_i' \cdot \left(-R_i'\right) = c - \sum_{i=1}^{t} L_i' R_i' - d = c - \sum_{i=1}^{t^2} L_i' R_i' - m$$

  which is equal to $c - a \times b - m$ (the rightmost equality follows from the analysis of the refresh gadget).
- **Multiplication and addition by constant gadgets.** Notice that these gadget do not use any masking inputs, and so the computation in these gadgets is always correct (corresponds to computation under the all-zeros attack).

References

1. Andrychowicz, M.: Efficient refreshing protocol for leakage-resilient storage based on the inner-product extractor. arXiv preprint arXiv:1209.4820 (2012)
2. Andrychowicz, M., Dziembowski, S., Faust, S.: Circuit compilers with $O(1/\log(n))$ leakage rate. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 586–615. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49896-5_21

3. Battistello, A., Coron, J.-S., Prouff, E., Zeitoun, R.: Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 23–39. Springer, Heidelberg (2016). doi:10.1007/978-3-662-53140-2_2
4. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: STOC 1988, pp. 1–10. ACM (1988)
5. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In: ITCS 2012, pp. 326–349 (2012)
6. Bitansky, N., Canetti, R., Goldwasser, S., Halevi, S., Kalai, Y.T., Rothblum, G.N.: Program obfuscation with leaky hardware. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 722–739. Springer, Heidelberg (2011). doi:10.1007/978-3-642-25385-0_39
7. Bitansky, N., Canetti, R., Halevi, S.: Leakage-tolerant interactive protocols. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 266–284. Springer, Heidelberg (2012). doi:10.1007/978-3-642-28914-9_15
8. Bitansky, N., Dachman-Soled, D., Lin, H.: Leakage-tolerant computation with input-independent preprocessing. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8617, pp. 146–163. Springer, Heidelberg (2014). doi:10.1007/978-3-662-44381-1_9
9. Boyle, E., Garg, S., Jain, A., Kalai, Y.T., Sahai, A.: Secure computation against adaptive auxiliary information. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 316–334. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40041-4_18
10. Boyle, E., Goldwasser, S., Jain, A., Kalai, Y.T.: Multiparty computation secure against continual memory leakage. In: STOC 2012, pp. 1235–1254 (2012)
11. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: FOCS 1988, pp. 11–19 (1988)
12. Cramer, R., Dodis, Y., Fehr, S., Padró, C., Wichs, D.: Detection of algebraic manipulation with applications to robust secret sharing and fuzzy extractors. In: Smart, N. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 471–488. Springer, Heidelberg (2008). doi:10.1007/978-3-540-78967-3_27
13. Dachman-Soled, D., Liu, F.-H., Zhou, H.-S.: Leakage-resilient circuits revisited – optimal number of computing components without leak-free hardware. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 131–158. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46803-6_5
14. Duc, A., Dziembowski, S., Faust, S.: Unifying leakage models: from probing attacks to noisy leakage. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 423–440. Springer, Heidelberg (2014). doi:10.1007/978-3-642-55220-5_24
15. Dziembowski, S., Faust, S.: Leakage-resilient circuits without computational assumptions. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 230–247. Springer, Heidelberg (2012). doi:10.1007/978-3-642-28914-9_13
16. Faust, S., Rabin, T., Reyzin, L., Tromer, E., Vaikuntanathan, V.: Protecting circuits from leakage: the computationally-bounded and noisy cases. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 135–156. Springer, Heidelberg (2010). doi:10.1007/978-3-642-13190-5_7

17. Genkin, D., Ishai, Y., Polychroniadou, A.: Efficient multi-party computation: from passive to active security via secure SIMD circuits. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 721–741. Springer, Heidelberg (2015). doi:10.1007/978-3-662-48000-7_35
18. Genkin, D., Ishai, Y., Prabhakaran, M., Sahai, A., Tromer, E.: Circuits resilient to additive attacks with applications to secure computation. In: STOC 2014, pp. 495–504 (2014)
19. Genkin, D., Ishai, Y., Weiss, M.: Binary AMD circuits from secure multiparty computation. In: Hirt, M., Smith, A. (eds.) TCC 2016. LNCS, vol. 9985, pp. 336–366. Springer, Heidelberg (2016). doi:10.1007/978-3-662-53641-4_14
20. Genkin, D., Ishai, Y., Weiss, M.: How to construct a leakage-resilient (stateless) trusted party. IACR Cryptology ePrint Archive (2017). http://eprint.iacr.org/2017/926
21. Goldreich, O.: The Foundations of Cryptography - Volume 1, Basic Techniques. Cambridge University Press, Cambridge (2001)
22. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: STOC 1987, pp. 218–229. ACM (1987)
23. Goldwasser, S., Rothblum, G.N.: Securing computation against continuous leakage. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 59–79. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14623-7_4
24. Goldwasser, S., Rothblum, G.N.: How to compute in the presence of leakage. In: FOCS 2012, pp. 31–40 (2012)
25. Goyal, V., Ishai, Y., Maji, H.K., Sahai, A., Sherstov, A.A.: Bounded-communication leakage resilience via parity-resilient circuits. In: FOCS 2016, pp. 1–10 (2016)
26. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003). doi:10.1007/978-3-540-45146-4_27
27. Ishai, Y., Weiss, M., Yang, G.: Making the best of a leaky situation: zero-knowledge PCPs from leakage-resilient circuits. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016. LNCS, vol. 9563, pp. 3–32. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49099-0_1
28. Juma, A., Vahlis, Y.: Protecting cryptographic keys against continual leakage. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 41–58. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14623-7_3
29. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). doi:10.1007/3-540-48405-1_25
30. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996). doi:10.1007/3-540-68697-5_9
31. Micali, S., Reyzin, L.: Physically observable cryptography. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 278–296. Springer, Heidelberg (2004). doi:10.1007/978-3-540-24638-1_16
32. Miles, E., Viola, E.: Shielding circuits with groups. In: STOC 2013, pp. 251–260 (2013)
33. Quisquater, J.-J., Samyde, D.: Electromagnetic analysis (EMA): measures and counter-measures for smart cards. In: Attali, I., Jensen, T. (eds.) E-smart 2001. LNCS, vol. 2140, pp. 200–210. Springer, Heidelberg (2001). doi:10.1007/3-540-45418-7_17

34. Rothblum, G.N.: How to compute under $\mathcal{AC}^0$ leakage without secure hardware. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 552–569. Springer, Heidelberg (2012). doi:10.1007/978-3-642-32009-5_32
35. Santis, A., Micali, S., Persiano, G.: Non-interactive zero-knowledge proof systems. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 52–72. Springer, Heidelberg (1988). doi:10.1007/3-540-48184-2_5
36. Weiss, M.: Secure computation and probabilistic checking. Ph.D. thesis (2016)
37. Yao, A.C.-C.: How to generate and exchange secrets (extended abstract). In: FOCS 1986, pp. 162–167 (1986)