Circuits Resilient to Additive Attacks with Applications to Secure Computation

Daniel Genkin
Technion
danielg3@cs.technion.ac.il

Yuval Ishai
Technion
yuvali@cs.technion.ac.il

Manoj M. Prabhakaran
University of Illinois
mmp@uiuc.edu

Amit Sahai
University of California
sahai@cs.ucla.edu

Eran Tromer
Tel Aviv University
tromer@cs.tau.ac.il

February 24, 2015

Abstract

We study the question of protecting arithmetic circuits against *additive* attacks, which can add an arbitrary fixed value to each wire in the circuit. This extends the notion of algebraic manipulation detection (AMD) codes, which protect *information* against additive attacks, to that of *AMD circuits* which protect *computation*.

We present a construction of such AMD circuits: any arithmetic circuit C over a finite field $\mathbb{F}$ can be converted into a functionally-equivalent randomized arithmetic circuit $\widehat{C}$ of size $O(|C|)$ that is fault-tolerant in the following sense. For any additive attack on the wires of $\widehat{C}$, its effect on the output of $\widehat{C}$ can be simulated, up to $O(|C|/|\mathbb{F}|)$ statistical distance, by an additive attack on just the input and output. Given a small tamper-proof encoder/decoder for AMD codes, the input and output can be protected as well.

We also give an alternative construction, applicable to small fields (for example, to protect Boolean circuits against wire-toggling attacks). It uses a small tamper-proof decoder to ensure that, except with negligible failure probability, either the output is correct or tampering is detected.

Our study of AMD circuits is motivated by simplifying and improving protocols for secure multiparty computation (MPC). Typically, securing MPC protocols against *active* adversaries is much more difficult than securing them against *passive* adversaries. We observe that in simple MPC protocols that were designed to protect circuit evaluation only against *passive* adversaries, the effect of any *active* adversary corresponds precisely to an additive attack on the original circuit's wires. Thus, to securely evaluate a circuit C in the presence of active adversaries, it suffices to apply the passive-secure protocol to $\widehat{C}$. We use this methodology to simplify feasibility results and attain efficiency improvements in several standard MPC models.

Contents

1 Introduction

1.1 Overview

The study of fault-tolerant circuits dates back to the work of von Neumann [vN56], who considered a model where every gate in a circuit can fail with some constant, and independent, probability. Subsequent works of Dobrushin and Ortyukov [DO77] and Pippenger [Pip85] showed how to construct fault-tolerant circuits in this model with only a logarithmic overhead in the worst case and a constant overhead in the typical case. Other models for fault-tolerant circuits, protecting against a bounded number of adversarial faults, were studied in [KLM94, GS95, GLM+04, IPSW06, FPV11, LL12, DK12, KLR12, DSK14].

In the present work we consider the goal of protecting boolean and arithmetic circuits against adversarial faults that may apply to *all* wires in the circuit. Even if one settles for *detecting* faults rather than fully protecting against faults, this goal would be too ambitious. Indeed, an attacker can simply rewrite the input or the output of the circuit without being detected. But there is a natural model, which is also motivated by the cryptographic applications discussed later, where achieving this goal is conceivable. In this model we limit the attacker in two ways:

1. The attacker cannot directly attack the input and the output to the circuit; instead, the input is fed to a small (randomized) tamper-proof input encoder and the output is obtained from a small tamper-proof output decoder.[1]

2. The class of attacks – i.e., mappings from the original wire values to the new wire values – is restricted.

Note that (1) alone is insufficient to remove the impossibility, since it does not rule out completely rewriting the output of the input encoder or the input to the output decoder, and (2) alone is insufficient since it does not rule out direct (albeit restricted) attacks on the input or output.

We instantiate (2) by considering *additive attacks*. That is, given a (possibly randomized) arithmetic circuit over a finite field $\mathbb{F}$, we allow an adversary to "blindly" add a field element of his choice to each wire in the circuit. In the case of boolean circuits, this amounts to toggling an arbitrary subset of the wires. Such additive attacks were previously considered in the context of error-correcting codes by Karpovsky and Nagvajara [KN89] and more recently by Cramer et al. [CDF+08], who constructed *algebraic manipulation detection* (AMD) codes which resist such attacks.[2] The main objective of the present work is to extend the notion of AMD codes, which protect *information* against additive attacks, to *AMD circuits*, which protect *computations* against such attacks.

We will start by defining a simpler notion of security against additive attacks (see Definition 1.1) that does not use any tamper-proof components (i.e., only the restriction (2) from above is used), but (inevitably) allows additive attacks on the input and output of the circuit. We show how to compile any arithmetic circuit C over a large finite field $\mathbb{F}$ into a functionally equivalent randomized arithmetic circuit $\widehat{C}$ of size $O(|C|)$ which is secure in this sense. The effect any additive attack has on the output of $\widehat{C}$ can be simulated, up to $O(|C|/|\mathbb{F}|)$ statistical distance, by applying a (randomized) additive attack to the input and output alone. Thus, as far as additive attacks are concerned, $\widehat{C}$ is essentially as good as a tamper-proof implementation of C in which only the input and output are exposed.

[1] By "small" we mean independent of the circuit complexity of the function being computed (but possibly depending polynomially on the input/output size). This rules out a trivial solution where the entire computation is carried out by tamper-proof hardware.

[2] In [CDF+08], algebraic manipulation detection codes were defined over an Abelian group, where the only manipulation allowed is an additive attack. We too are considering additive attacks, but since we work over a field (which contains a multiplication operation as well), a more appropriate term in our context would be "*Additive* Manipulation Detection" codes. Indeed, allowing the attacker full *algebraic* manipulation of the field elements, or even just affine attacks, would let it circumvent the security guarantee of the AMD codes of [CDF+08].

Combining the above construction with small tamper-proof encoder and decoder for AMD codes, the input and output can be protected as well. That is, any arithmetic circuit C over a large finite field can be compiled into a functionally equivalent randomized circuit of comparable size that uses small tamper-proof input encoder and output decoder, and is guaranteed to either produce the correct output of C or set an error flag, except with negligible failure probability. This construction has an additional security feature that will be useful for our motivating applications: Even in the presence of an additive attack, the field elements fed into the output decoder (and in particular the final output) reveal essentially nothing about the input x beyond $C(x)$.

The above construction offers no security guarantees when $\mathbb{F}$ is small. For the general case we present a more complex construction which uses small tamper-proof encoder and decoder to ensure that, except with negligible failure probability, either the output is correct or tampering is detected. More concretely, to achieve $2^{-\sigma}$ error probability, the size of the AMD circuit $\widehat{C}$ is $|C| \cdot \mathrm{poly}(\sigma)$. This construction can be used for protecting boolean circuits against wire-toggling attacks. However, here we do not realize the stronger security feature discussed above.

Cryptographic applications of AMD circuits. Our study of AMD circuits is further motivated by observing that they are useful for the design of protocols for *secure multiparty computation* (MPC). An MPC protocol allows two or more mutually distrusting parties to perform a distributed computation on their local inputs without compromising the secrecy of the inputs or the correctness of the outputs. Following the seminal works from the 1980s that established the general *feasibility* of secure computation [Yao86, GMW87, BGW88, CCD88, RB89], significant research efforts have been invested into studying *efficiency* questions in this area.

It is typically much easier to secure MPC protocols against *passive* adversaries, who may try to learn information about secret inputs but do not otherwise deviate from the protocol, than against *active* adversaries who may arbitrarily deviate from the protocol. The security of protocols that were only designed to withstand passive attacks may break down completely if the adversary is active. While there are general techniques for strengthening security against passive attacks into security against active attacks (most notably, the "GMW paradigm" [GMW87]), these involve a considerable overhead and do not apply at all to the type of protocols considered here.

Our key observation is that in natural MPC protocols that offer information-theoretic security against passive attacks, any cheating strategy of an active adversary can be modeled as an *additive* attack on the underlying circuit that the parties are trying to compute. This holds both for protocols in the setting of an honest majority, such as the "BGW protocol" [BGW88] and its more efficient variant from [DN07], and for protocols in the setting of no honest majority, such as variants of the "GMW protocol" over an ideal oblivious transfer oracle [GMW87, Gol04] or an OLE oracle[3] [IPS09].

The above observation gives rise to a novel methodology for the design of MPC protocols with security against active adversaries. Instead of designing a complex protocol for evaluating f that explicitly protects against active attacks, apply a simple protocol, which was only designed to protect against passive attacks, to evaluate an AMD circuit for f. (The role of the input encoder and the output decoder can be emulated via local computation and does not require interaction.) Thus, the most challenging aspect of MPC protocol design is reduced to the arguably cleaner problem of AMD circuit design.

We demonstrate the usefulness of this methodology by applying it to simplify and improve on previous results in the area of MPC.[4] For instance, we can derive the feasibility of active-secure MPC in the presence of an honest majority [RB89] from the much simpler passive-secure BGW protocol [BGW88],

[3] An OLE oracle receives $a, b \in \mathbb{F}$ from one party and $x \in \mathbb{F}$ from another, and returns $ax + b$ to the latter. OLE can be viewed as an arithmetic generalization of oblivious transfer.

[4] Here and in the following, we allow the adversary to abort the computation and do not attempt to guarantee output delivery. This is often inevitable, e.g., when there is no honest majority, or there is an honest majority but no broadcast channel, or in security models where the adversary can block messages exchanged between honest parties (cf. [Can01]).

as well as the feasibility of active-secure MPC protocols with no honest majority [GMW87, Kil88, IPS08, IPS09] (given an OLE oracle) from their much simpler passive-secure counterparts. We also obtain a new feasibility result for MPC with no honest majority using a corruptible source of correlated randomness. On the efficiency front, we apply our methodology to a simplified variant of a passive-secure protocol from [DN07] to obtain a simpler and more efficient alternative to a recent protocol from [BFO12]. We also obtain the first active-secure two-party protocol for evaluating an arbitrary arithmetic circuit over a large field using only a constant number of calls to an OLE oracle for each gate in the circuit.

1.2 Our contribution

We now give a more detailed outline of our results. In Section 1.2.1 we summarize results on protecting circuits against additive attacks and in Section 1.2.2 we summarize the applications to secure multiparty computation.

1.2.1 Protecting circuits against additive attacks

We start by defining our main notion of security with respect to additive attacks. Let $f : \mathbb{F}^n \to \mathbb{F}^k$ be a function to be computed. We say that a randomized arithmetic circuit[5] $\widehat{C} : \mathbb{F}^n \to \mathbb{F}^k$ is an ϵ-secure implementation of f if $\widehat{C}$ correctly computes f when it is not attacked, and moreover any additive attack on $\widehat{C}$ has the same effect on the output of $\widehat{C}$ (up to an ϵ statistical error) as applying some additive attack to the inputs and outputs alone:

Definition 1.1 (Additive-attack security)**.** *A randomized circuit $\widehat{C} : \mathbb{F}^n \to \mathbb{F}^k$ is an* ϵ-secure implementation *of a function $f : \mathbb{F}^n \to \mathbb{F}^k$ if the following holds:*

- Completeness. *For all $\mathbf{x} \in \mathbb{F}^n$ it holds that* $\Pr[\widehat{C}(\mathbf{x}) = f(\mathbf{x})] = 1$.
- Additive-attack security. *For any circuit $\widetilde{C}$ obtained by subjecting $\widehat{C}$ to an additive attack, there exists $\mathbf{a}^{\mathsf{in}} \in \mathbb{F}^n$ and a distribution $\mathcal{A}^{\mathsf{out}}$ over $\mathbb{F}^k$ such that for any $\mathbf{x} \in \mathbb{F}^n$ it holds that*

$$SD\left(\widetilde{C}(\mathbf{x}), f(\mathbf{x} + \mathbf{a}^{\mathsf{in}}) + \mathcal{A}^{\mathsf{out}}\right) \leq \epsilon,$$

  where SD denotes statistical distance between two distributions.

We naturally extend the definition to the case where $f : \mathbb{F}^n \to \mathbb{F}^k$ is a randomized *function. In this case completeness requires that the output distribution of $\widehat{C}(\mathbf{x})$ and $f(\mathbf{x})$ be identical. Finally,we say that $\widehat{C}$ is an ϵ-secure implementation of a circuit C if $\widehat{C}$ is an ϵ-secure implementation of the (possibly randomized) function f computed by C.*

In Sections 3.2 and 3.3 we prove that every circuit C over a large finite field can be compiled into a circuit $\widehat{C}$ that is secure against additive attacks. Formally, we prove the following theorem.

Theorem 1.1 (see Theorem 3.7)**.** *For any field $\mathbb{F}$ and (possibly randomized) arithmetic circuit $C : \mathbb{F}^n \to \mathbb{F}^k$ there exists a randomized circuit $\widehat{C} : \mathbb{F}^n \to \mathbb{F}^k$ such that $\widehat{C}$ is an ϵ-secure implementation of C where $\epsilon = O(|C|/|\mathbb{F}|)$. Moreover, $|\widehat{C}| = O(|C|)$.*

The notion of additive-attack security in Definition 1.1 above still allows for an attack on the inputs and outputs of the circuit. This is because the adversary is allowed to attack every wire in the circuit, and in particular input and output wires. Thus, we need a randomized, *tamper-proof* input encoder Enc and output decoder Dec in order to prevent attacks against the inputs and outputs. We would like

[5] An arithmetic circuit consists of field addition, subtraction, and multiplication gates. If it is randomized, it may also include randomness gates that output uniformly random field elements. We write $\widehat{C} : \mathbb{F}^n \to \mathbb{F}^k$ to indicate that the input of $\widehat{C}$ consists of n field elements (not including randomness gates) and its output consists of k field elements.

the size of Enc and Dec to be kept as small as possible (and in particular much smaller than the circuit being computed).

Notice that even in the presence of a decoder that cannot be attacked, the adversary is still allowed to attack all the wires leading from the circuit to the decoder. Thus, we cannot hope for *correcting* the result following an additive attack but must settle for a weaker guarantee of *detecting* the attack. We capture this by allowing Dec to have a special output, denoted *flag*, where if this output is nonzero this means that an attack has been detected.

The circuits Enc and Dec will perform an AMD encoding of the input and an AMD decoding of the output, respectively. (See Section 2.2 for a formal definition and constructions of AMD codes.) The circuit $\widehat{C}$, which gets input from Enc and produces output for Dec, is obtained by applying Theorem 1.1 to the circuit C' obtained from C by applying an AMD decoder to its input and an AMD encoder to its output. See Theorem 3.8 for a formal statement of the result we get by applying this construction to Theorem 1.1.

Theorem 1.1 does not provide any security guarantees for circuits over small fields. In particular, it cannot be used to protect boolean circuits. To handle general fields, we need to rely on a small, tamper-proof output decoder. Moreover, unlike the previous construction, here we only guarantee the *correctness* of the output(unless an error is detected) and do not provide any guarantees regarding the secrecy of the input in the presence of additive attacks. Below we define the stronger notion of correctness, which does not require a tamper-proof input encoder. (As before, the input can be protected by an input encoder via the use of AMD codes.) This feature will be useful when applying a composition-based approach for constructing AMD circuits in this setting.

Definition 1.2 (Additive-attack correctness with decoder)**.** *Let $\mathbb{F}$ be a finite field and let $f : \mathbb{F}^n \to \mathbb{F}^k$. We say that a pair of circuits $(\widehat{C}, D)$ are an ϵ-correct implementation of f with a decoder if the following holds:*

- Completeness. *For all $\mathbf{x} \in \mathbb{F}^n$, we have $\Pr[D(\widehat{C}(\mathbf{x})) = (0, f(\mathbf{x}))] = 1$.*
- Additive-attack correctness. *For any circuit $\widetilde{C}$ obtained by subjecting $\widehat{C}$ to an additive attack there exists $\mathbf{a}^{\mathsf{in}} \in \mathbb{F}^n$ such that for all $\mathbf{x} \in \mathbb{F}^n$*

$$\Pr\left[D(\widetilde{C}(\mathbf{x})) \notin \mathsf{ERR} \cup \{(0, f(\mathbf{x} + \mathbf{a}^{\mathsf{in}}))\}\right] \leq \epsilon$$

  where $\mathsf{ERR} = \{(z', z) : z' \in \mathbb{F} \setminus \{0\}, z \in \mathbb{F}^k\}$ and the probability is taken over the internal randomness of $\widetilde{C}$.

Like in Definition 1.1, we say that $(\widehat{C}, D)$ is an ϵ-correct implementation of a (deterministic or randomized) circuit C if $(\widehat{C}, D)$ is an ϵ-correct implementation of the function f computed by C.

In Section 4.2 we prove the following theorem.

Theorem 1.2 (see Theorem 4.5)**.** *For any field $\mathbb{F}$, positive integer σ and arithmetic circuit $C : \mathbb{F}^n \to \mathbb{F}^k$ there exist $(\widehat{C}, D)$ that form an ϵ-correct implementation of C with a decoder, where $\epsilon = 2^{-\sigma} \cdot |C|$, $|\widehat{C}| = |C| \cdot poly(\sigma)$, and $|D| = k \cdot poly(\sigma)$.*

Notice the differences between Theorems 1.1 and 1.2 above. Theorem 1.1 guarantees security for arithmetic circuits over large fields while Theorem 1.2 achieves the weaker notion of additive-attack correctness, which allows information to leak via the error flag, but without requiring the underlying field to be large. In particular, Theorem 1.2 can be used over the binary field.

Finally, in Appendix A we present an alternative construction based on a *Multiprover Interactive Proof* (MIP) systems. The parameters that can be obtained from known MIP systems are summarized by the following theorem.

Theorem 1.3 (see Theorem A.2)**.** *For any positive integer σ and for any circuit descriptor F describing a circuit $C : \{0,1\}^n \to \{0,1\}^k$ there exists a $2^{-\sigma}$-correct implementation $(\widehat{C}, D)$ of C such that $|\widehat{C}| = (\sigma + |C| \cdot |F|) \cdot polylog(\sigma, |C|, |F|)$ and $|D| = (n + k + |F|) \cdot \sigma \cdot polylog(|C|, |F|)$.*

The advantage of the above construction over the previous one is that, for succinctly described circuits (see Definition A.1), the multiplicative overhead to the circuit size is only polylogarithmic in the security parameter σ. On the downside, the construction of Theorem 1.3 requires the decoder D to grow with the description size of C, thus making it meaningful only for circuits that have a succinct description. Moreover, D needs to grow linearly with the input size of C. Finally, this approach does not produce a construction with full additive-attack security, but only additive-attack correctness.

1.2.2 Multiparty computation via AMD circuits

The notion of AMD circuits is motivated by the following application to secure multiparty computation (MPC). Our goal is to construct MPC protocols that are secure against active adversaries, starting from those which are secure only against passive adversaries. Unlike the prevalent approach of modifying the protocol itself to directly handle any deviations of an active adversary, our approach is to use the protocol as it is, but apply it to a modified circuit. That is, given an MPC protocol, secure against passive adversaries, for a function f computed by a circuit C, we apply the *same* protocol to a modified circuit $\widehat{C}^{\mathsf{AMD}}$. The circuit $\widehat{C}^{\mathsf{AMD}}$, in addition to computing the function, is also responsible for handling any consequences resulting from the adversary's deviations from the protocol. The circuit $\widehat{C}^{\mathsf{AMD}}$ will be essentially an additively secure version of the original circuit; we show that, for several simple MPC protocols from the literature that were only designed to provide security against passive adversaries, this approach suffices to handle general active adversaries. This is discussed in Section 5.

In the following we describe different applications of this methodology in the context of prior results (see summary in Table 1).

For simplicity we consider an MPC model where the adversary can abort the execution of the protocol, and do not attempt to provide guaranteed output delivery. The latter can be achieved when there is an honest majority and a broadcast channel is available [RB89].[6] We note, however, that protecting against active attacks is highly nontrivial even in this setting, and moreover the efficiency comparison with previous works takes this simpler model into account.

We begin by deriving a simple version of the feasibility result of [RB89], for MPC in the presence of an honest majority, from the passive-secure BGW protocol with $n = 2t + 1$ parties. Formally, we obtain the following theorem (see Section 5.7):

Theorem 1.4 (see Corollary 5.4)**.** *For any n-party functionality f represented by an arithmetic circuit C over a sufficiently large $\mathbb{F}$ there exists an n-party protocol π that ϵ-securely computes f with abort in the presence of an honest majority for $\epsilon = O(|C|/|\mathbb{F}|)$. The protocol involves communication of $O(n^2|C|)$ field elements.*

Here and in the following, one can eliminate the dependence of the error on the field size by using an extension field. This results in a multiplicative overhead of at most σ for reducing the error to $2^{-\sigma}$ (see Corollaries 5.5, 5.8, 5.9 and 5.11).

In Section 5.8 we obtain a more efficient variant which has a communication complexity of $O(n|C| + n^2)$ field elements. This asymptotically matches the communication complexity of the best known *passive-secure* protocol from [DN07], and is obtained by applying our methodology to this protocol.

[6] Our protocols can be modified to have this feature, whenever it is achievable, by using standard techniques; however, the details are beyond the scope of this work.

Adv	Resilience	Security	Communication complexity	Model	Ref
passive	$\|T\| < n/2$	perfect	$O(n^2\|C\|)$	plain	[BGW88]
passive	$\|T\| < n/2$	perfect	$O(n\|C\| + n^2)$	plain	[DN07]
passive	$\|T\| < n$	perfect	$O(n^2\|C\|)$ for boolean circuits	OT	[GMW87]
passive	$\|T\| < n$	perfect	$O(n^2\|C\|)$	OLE	[IPS09]
active	$\|T\| < n/2$	statistical	$\text{poly}(n) \cdot \|C\|$	plain	[RB89]
active	$\|T\| < n/2$	statistical	$O(n\|C\| + n^2 \log n \cdot d_C) + \text{poly}(n)$	plain	[BFO12]
active	$\boldsymbol{\|T\| < n/2}$	**statistical**	$\boldsymbol{O(n^2\|C\|)}$	**plain**	**§5.7**
active	$\boldsymbol{\|T\| < n/2}$	**statistical**	$\boldsymbol{O(n\|C\| + n^2)}$	**plain**	**§5.8**
active	$\|T\| < n$	statistical	$O(n^2\|C\| + \log \|\mathbb{F}\| \cdot d_C)$	OT+OLE	[IPS09]
active	$\|T\| < (1/2 - \epsilon)n$	statistical	$O(\log n \cdot \|C\|) + \text{poly}(n, d_C)$	plain	[DIK10]
active	$\boldsymbol{\|T\| < n}$	**statistical**	$\boldsymbol{O(n^2\|C\|)}$	**OLE**	**§5.9**
active	$\boldsymbol{\|T\| < n}$ **or** $\boldsymbol{T = \{\text{dealer}\}}$	**statistical**	$\boldsymbol{O(n^2\|C\|)}$	**plain**	**§5.10**

Table 1: Comparison of information-theoretic MPC protocols for arithmetic circuits. In the above, n is the number of parties, ϵ is an arbitrary small positive constant, C is an arithmetic circuit over a finite field $\mathbb{F}$, d_C is the multiplicative depth of C, and T is the set of parties corrupted by the adversary. Statistical security means that the protocol securely realizes C (with abort) with at most $O(|C|/|\mathbb{F}|)$ simulation error. The communication complexity column counts the total number of field elements that are communicated between the parties (where in the plain model we assume only the availability of secure point-to-point channels). An OLE oracle (an arithmetic generalization of OT) receives $a, b \in \mathbb{F}$ from one party and x from another, and returns $ax + b$ to the latter. The results highlighted in boldface are new.

Theorem 1.5 (see Corollary 5.7)**.** *For any n-party functionality f represented by an arithmetic circuit C over a sufficiently large $\mathbb{F}$ there exists an n party protocol π with communication complexity of $O(n|C| + n^2)$ field elements, where π ϵ-securely computes f with abort in the presence of an honest majority for $\epsilon = O(|C|/|\mathbb{F}|)$.*

This gives a simpler alternative to a recent protocol from [BFO12] and improves its complexity by eliminating a quadratic overhead for each layer of the circuit, as well as a large polynomial additive term. See Table 1.

Next, we tackle the task of secure multiparty computation in the presence of an active adversary without an honest majority. Unfortunately, this task is impossible to achieve for arbitrary circuits in the plain model. Thus, we are forced to use some kind of a hybrid model or have an honestly-executed input-independent preprocessing phase which is done before the execution of the protocol. In Section 5.9 we present results for secure multiparty computation using an arithmetic generalization of the OT-hybrid model, called the OLE-hybrid model [NP06, IPS09] (where an oracle receives $a, b \in \mathbb{F}$ from one party and x from another, and returns $ax + b$ to the latter) and in Section 5.10 we present our results in the preprocessing model.

Concretely, in Section 5.9 we use an arithmetic version of the GMW protocol [GMW87, IPS09] and obtain an n-party protocol for securely computing any functionality (represented by an arithmetic circuit C), without requiring an honest majority, using $O(n^2|C|)$ calls to the OLE oracle. This improves over the protocol of [IPS09] that inherently requires $\Omega(\sigma)$ additional oracle calls for achieving $2^{-\sigma}$-security, regardless of the field or circuit size. Formally:

Theorem 1.6 (see Theorem 5.7)**.** *For any n-party functionality f represented by an arithmetic circuit*

C there exists a protocol π in the OLE-hybrid model that ϵ-securely computes C with abort for $\epsilon = O(|C|/|\mathbb{F}|)$. Moreover, π invokes the OLE oracle $O(|C|n^2)$ times.

Finally, in Section 5.10 we address the goal of secure multiparty computation in the preprocessing model. We present a protocol for securely computing an n-party functionality (again represented as an arithmetic circuit C) that utilizes a preprocessing phase which runs before the computation of f starts and does not depend on the inputs to f. This phase can be implemented using an additional party called the *dealer* that sends correlated randomness to the parties. We strengthen previous results in the preprocessing model [IPS09, BDOZ11, IKM+13], which assume the dealer to be honest, by providing security when either the dealer or any subset of the other parties may be corrupted (though not both). Formally:

Theorem 1.7 (see Theorem 5.9)**.** *For any n-party functionality represented by an arithmetic circuit C over $\mathbb{F}$, there exists a protocol π that uses an additional dealer, such that π is an ϵ-secure protocol for computing C with abort against an active adversary controlling either the dealer or any other parties, where $\epsilon = O(|C|/|\mathbb{F}|)$. The dealer in π only distributes correlated randomness to the other parties.*

1.3 Overview of techniques

1.3.1 Additive-attack security

We first present our result for additive-attack security (see Section 3.2 for details) for a computation over a large field $\mathbb{F}$. Suppose we are given an arithmetic circuit C. In its secure version $\widehat{C}$, every wire of C is paired with a wire that carries a "MAC tag." Each gate in C is replaced by a small gadget which computes the original gate's output as well as a MAC tag for it; further, this gadget accepts the MAC tags of the inputs to the original gate, and carries out a MAC verification computation. Note that this verification circuitry itself is open to additive attacks. Nevertheless, we can arrange that $\widehat{C}$ will produce a random output if the MAC tag verification fails for a wire anywhere in the computation.

We present more details below. In the following we identify the gate g with its result; the meaning will be clear from the context.

The basic construction. For a gate's output wire g, its MAC value will simply be $g \cdot v^d$, where v is a randomly generated element of the underlying field $\mathbb{F}$ (fixed to the same value for all gates), and d is the degree of the wire g (as a polynomial in the input variables). This MAC has property that multiplications can be performed on the MAC value homomorphically to obtain a value that can correspond to a MAC value of the result of a multiplication. Updating the MAC for an addition and subtraction gate is implemented using a simple gadget. The consistency check is implemented as follows. We first compute the result of the original gate, g. Next, we compute the MAC value in two ways. The first way is by directly multiplying g to v^d (in turn computed from v). The second way is using the MAC values of the inputs (homomorphically for multiplication and via a simple gadget for the case of addition and subtraction). We then check that the values are equal: more precisely, in each gate we take the difference of these two values, and linearly combine them across all gates using random coefficients; the result – which is a random field element if any inconsistency was detected, and 0 otherwise – is added to the final outcome.

We show that any additive attack on $\widehat{C}$ is either equivalent to an additive attack on the input wires and /or output wires only, or else, results in the output being random, up to a statistical distance of $O(d/|\mathbb{F}|)$ from the uniform distribution. Note that if the field is large (i.e., is of size exponential in the security parameter) and if d is small (for e.g., polynomial), we obtain negligible error in security. The security of the construction requires that the underlying circuit C be such that for every gate in C, the joint values of its two inputs should be almost uniformly distributed over $\mathbb{F} \times \mathbb{F}$. This is ensured by first compiling C into an appropriately randomized circuit (see below).

One problem with the above basic construction is that the security error grows with the degree of the circuit. Since the degree of a circuit can be exponential in its depth, this construction does not yield a full solution to our problem. However, we show that bootstrapping from this construction for low-degree circuits, we can indeed obtain a construction that is secure for all polynomial-sized circuits (see below).

The randomization process. As noted above, the basic construction relies on the inputs to each gate of the given circuit being uniformly random. We can enforce this as follows. Each wire a inside the circuit C will be replaced by two wires, carrying values $a + r_1$ and $a + r_2$, where the masking values r_1 and r_2 are generated as random field elements (that are the same throughout the circuit). Next, we will replace each multiplication and addition gate with a gadget that will get as input $(a + r_1, a + r_2, b + r_1, b + r_2)$ and output either $(ab + r_1, ab + r_2)$, $(a + b + r_1, a + b + r_2)$ or $(a - b + r_1, a - b + r_2)$ respectively. These gadgets have the property that the inputs of every internal gate are completely random. To complete the modification of the circuit, two additional layers are added. First, a layer of addition gates is added to the input wires to carry out the encoding. Next, a layer of subtraction gates is added to the output wires to carry out the decoding. Note that the inputs to the gates in these additional layers do not have the randomness property we set out to ensure for every gate (since the inputs and outputs are not random). However, attacks on these addition and subtraction gates are equivalent to attacks on the actual input and output of the circuit itself, and this is permitted by Definition 1.1.

From low-degree circuits to arbitrary circuits. Observe that additive-attack security could be easily achieved if we were allowed to use tamper-proof gadgets to implement each gate. Then each gate can be replaced by a tamper-proof component that gets two inputs encoded using an AMD code and, after decoding them, computes an AMD encoding of the gate's result. In our final construction, we implement these gadgets using the above construction for low-degree circuits and obtain a construction for arbitrary circuits.

1.3.2 Handling small fields

Our constructions for additive-attack security inherently fail when the underlying field is small, even if we were willing to tolerate a small constant error (see Section 3.2.4 for details). In this section, we present an alternative construction that achieves additive-attack *correctness* over small fields, with negligible error (see Section 4.2 for details). Recall that correctness prevents the attacker from causing the circuit to output a wrong value without being detected.

Basic construction without a decoder. Our final construction will achieve additive-attack correctness using a small tamper-proof output decoder. But first, we present a construction that does not use any decoders but allows (inevitably) both inputs and outputs to besubject to additive attacks (see Definition 4.1). This construction will have a constant error. Later we will show how to amplify the correctness (and also improve the efficiency) of this construction, and meet the requirements of Definition 1.2, by relying on a small tamper-proof output decoder.

The basic idea is that our new circuit would compute not only the output of the original circuit, but also a *proof* that the output is correct; at the end of the computation, this proof will be verified by another part of the circuit. We need a simple proof system that can be implemented in such a way that soundness holds even when *the verifier as well as the prover could be under (additive) attack.*[7] Our proof system follows in the pattern of the Hadamard PCP system of [ALM+92], which turns out to have the linearity properties suitable for our purposes. However, we cannot use this PCP system as it is, since the proof is exponentially large. We use an alternate compact representation of the proof that suffices *if* we can ensure that the prover indeed computes prescribed linear functions of a purported

[7] Even if we allowed a small tamper-proof decoder (which we do not for this basic construction), it would not be feasible to house the verifier there, since the verifier would be at least as large as the original circuit itself; allowing such a large tamper-proof component trivializes the problem.

witness and the verifier's queries. This condition on the prover is enforced by a "matrix multiplication gadget" (see below). The verifier's computation is simple and results in an error flag to be set (to a non-zero value) with at least a constant probability, if the proof is not valid.

It remains to ensure that under additive attack, the prover is restricted to computing the correct linear functions (but possibly using an invalid witness). This is achieved using the following gadgets.

The multiplication gadgets. We sketch our ϵ-correct implementation (without output decoder) of a matrix-by-vector multiplication. For this, first we construct a scalar-by-vector multiplication gadget.

The inputs to a scalar-by-vector multiplication gadget consist of a vector $\mathbf{v}$ and a scalar x, the output is $\mathbf{v}x$. The main idea of this construction is to make the circuit compute $\mathbf{z} = \mathbf{v}x$ and $q' = (\mathbf{r} \cdot \mathbf{v})\, x$ where $\mathbf{r}$ is a random vector. To verify that $\mathbf{r} \cdot \mathbf{z} = q'$, we compute $f = \mathbf{r}\mathbf{z} - q'$ as an error flag. We show that any attack that does not correspond to an additive attack on the inputs and outputs of the scalar-by-vector multiplication gadget will cause the flag to be set randomly.

We next proceed to the matrix-by-vector multiplication gadget. The inputs to such a gadget are a matrix M and a vector $\mathbf{x}$, and the output is $\mathbf{z} = M\mathbf{x}$. We implement this gadget using the scalar-by-vector multiplication gadget. The main idea is as follows: we treat the *columns* of M as vectors and multiply each column with the required coordinate of $\mathbf{x}$ using the scalar-by-vector multiplication gadget. Afterwards, we sum up these intermediate values to obtain the output of the matrix-by-vector multiplication. Since any attack on the scalar-by-vector multiplication gadget is equivalent to an attack on its inputs and outputs and since the matrix-by-vector multiplication gadget only sums up the results of the scalar-by-vector multiplication gadget, it will be the case that any attack on the matrix-by-vector multiplication gadget is either equivalent to an attack on its inputs and outputs or it causes its error flag to become non-zero with constant probability.

Theorem 1.8 (informal; see Lemma 4.2)**.** *For any finite field $\mathbb{F}$, the function $f(M, \mathbf{v}) = M\mathbf{v}$ admits a* 0.95*-correct implementation $\widehat{C}$.*

Using this gadget to implement the prover in the above outline, we obtain the following result.

Theorem 1.9 (informal; see Theorem 4.1)**.** *Any circuit C over a finite field $\mathbb{F}$ admits a* 0.997*-correct implementation without output decoder, $\widehat{C}$, where $|\widehat{C}| = O(|C|^2)$.*

Correctness amplification. For small fields (in particular, the binary field), the above construction has a high error probability. A naive attempt at reducing the error to ϵ^σ would be to repeat the ϵ-correct construction σ times, and then use a (tamper-proof) decoder to check for consistency. However, it is possible that different instances will be operating on different inputs, and therefore no amplification will be achieved (see Section 4.2.1). This problem can be solved by asking each instance of the construction to output its input in addition to the result and then using a decoder to verify that all the inputs are consistent. However, in this case the complexity of the decoder will be polynomial in the input size. Keeping the tamper-proof decoder size virtually independent (up to logarithmic factors) of the input size is crucial for the efficiency improvement we discuss next. Thus, we use a family of (almost pairwise independent) hash functions such that the circuit will output a hash digest of its input instead of the actual input. Since input consistency is still verified, attacks that cause different instances of the construction to operate on different inputs will cause inconsistency in the hash digests, and the decoder will then set the error flag wire to be non-zero.

Theorem 1.10 (informal; see Theorem 4.3)**.** *Any circuit $C : \mathbb{F}^n \to \mathbb{F}^k$ admits a $2^{-\sigma}$-correct implementation $(\widehat{C}, D)$ where $|\widehat{C}| = |C|^2 \cdot poly(\sigma)$ and $|D| = k \cdot poly(\sigma)$.*

From quadratic to linear overhead. The above construction has quadratic overhead in the circuit size, since we use parts from the PCP prover of [ALM+92]. In particular, similarly to [ALM+92], our construction will compute all possible multiplications of two intermediate wires inside the circuit. We

improve this using "bootstrapping", as follows. We go over the gates of C in topological order. For each input gate, we apply the above construction to the single-wire identity circuit, yielding an ϵ-correct gadget, and a corresponding decoder. Then, for each subsequent gate g, we consider the small circuit C' consisting of the decoders corresponding to the two gates of upstream of g, along with g itself, and apply the above construction to C' to yield an ϵ-correct gadget and a new decoder, and so on. These are wired together. Finally, the decoders corresponding to the output gates, taken together, are considered the decoder for the resulting ϵ-correct implementation of C.

Since the substitution replaces a gate with a small gadget whose size is independent of C, the resulting circuit size grows linearly with that of $|C|$.

Theorem 1.11 (informal; see Theorem 4.5)**.** *Any circuit $C : \mathbb{F}^n \to \mathbb{F}^k$ admits a $(2^{-\sigma} \cdot |C|)$-correct implementation $(\widehat{C}, D)$ where $|\widehat{C}| = |C| \cdot poly(\sigma)$ and $|D| = k \cdot poly(\sigma)$.*

1.3.3 Secure multiparty computation

We review the main techniques used for applying AMD circuits towards secure computation in the presence of an active adversary, as discussed in Section 1.2.2. (see Section 5)

Protecting the computation of circuits. We start from a protocol π that evaluates a circuit C with security against *passive* adversaries. In Section 5 we prove, for several useful protocols π, that when π is executed in the presence of an *active* adversary, π actually computes a circuit $\widetilde{C}$ that is the same as C up to some additive attack that is chosen by the adversary. Thus, by replacing the circuit C with an additive-attack secure implementation $\widehat{C}$ of C we obtain that any active attack on the protocol corresponds to an additive attack on the inputs and outputs of C.

Protecting the inputs and outputs. To protect the inputs and outputs of C against additive attacks, we construct another circuit C^{AMD} from C so that C^{AMD} gets its inputs in some AMD code, decodes them, and then applies C. Finally, C^{AMD} encodes the outputs of C using an AMD code. In addition, if C^{AMD} gets inputs that are not valid AMD encodings due to an additive attack by the adversary, C^{AMD} sets a special output flag to be random. This will notify the honest parties that they should abort the computation since the results might have been corrupted by the adversary.

The final protocol. We construct an active-secure MPC protocol π' for C as follows. First, all the parties locally encode their inputs using an AMD code. Then they invoke π on an additive-attack secure implementation $\widehat{C}^{\mathsf{AMD}}$ of C^{AMD}. Finally, the parties locally decode the outputs of C^{AMD} obtained from the execution of π and abort if the decoding fails or if the error flag is nonzero. The security of π' is argued as follows. Notice that by the properties of π, the adversary is limited to only performing additive attacks on $\widehat{C}^{\mathsf{AMD}}$. Since $\widehat{C}^{\mathsf{AMD}}$ is additive-attack secure, these attacks are equivalent (up to small statistical distance) to additive attacks on the inputs and outputs of C^{AMD}. Finally, notice that any additive attack on the inputs and outputs of $\widehat{C}^{\mathsf{AMD}}$ will be detected by the AMD code, causing the honest parties to abort.

Applications to existing semi-honest MPC protocols. While our approach can be possibly extended to include many other protocols, in Section 5 we demonstrate our approach by using variants of existing semi-honest MPC protocols. Namely a variant of the semi-honest BGW protocol [BGW88], an arithmetic version of the semi-honest GMW protocol in the OLE-hybrid model [GMW87, IPS09] and a simplified version of the semi-honest DN protocol [DN07]. Thus, we are able to obtain protocols that are secure against active adversaries that have asymptotically the same communication complexity and number of oracle calls as their semi-honest versions.

Applying our methodology to the semi-honest DN protocol we obtain an n-party protocol π for $O(|C|/|\mathbb{F}|)$-securely computing a circuit C over some field $\mathbb{F}$ with abort in the presences of an honest majority whose communication complexity is $O(n|C| + n^2)$ field elements. Similarly, applying our methodology arithmetic version of the semi-honest GMW protocol in the OLE hybrid model [GMW87,

IPS09] we obtain an n-party protocol π for $O(|C|/|\mathbb{F}|)$-securely computing a circuit C over some field $\mathbb{F}$ in the OLE hybrid model that invokes the OLE oracle $O(n^2|C|)$ times.

Next, replacing the OLE oracle in the above protocol with a random OLE oracle (while making the necessary modifications to the protocol) and then asking an additional party called the *dealer* to compute the outputs of the random OLE oracle for all the parties, we obtain a protocol for computing a circuit C that is remains $O(|C|/|\mathbb{F}|)$-secure even when the adversary is allowed to corrupt either the dealer or any subset of the other parties (but not both).

1.4 Related work

Fault tolerant circuits. The goal of securing cryptographic hardware against active attacks has motivated different models for fault-tolerant circuits that mainly aim to protect the *secrecy* of the data stored inside the circuits. All prior works along this line somehow restrict the attacker so that some of the wires in the circuit are unaffected. This could be done by either restricting the number of attacked wires or by requiring that the attack fail with some probability. In our case, we eliminate this requirement by only considering the restricted class of additive attacks.

Gennaro et al. [GLM+04] and, more recently, Tauman-Kalai et al. [KKS11] considered tampering attacks that apply only to the *memory* but not to the circuit logic. The work of Liu and Lysyanskaya [LL12] considered the question of protecting circuits against leakage and tampering in the split-state model, where the leakage and tampering functions are not allowed to operate on the entire circuit at once but only on different parts of it. Ishai et al. [IPSW06], as well Dachman-Soled and Tauman-Kalai [DK12, DSK14], considered a reactive setting where in each clock cycle, the circuit produces outputs as well as updates its internal state. In their model, no part of the circuit must be free from tampering, but the adversary is restricted to tampering with a *bounded* number of wires in each clock cycle. Finally, Faust et al. [FPV11] considered a variant in which the adversary can attack every wire in the circuit, but each attack fails with some constant probability.

Additive attacks. As noted above, the goal of protecting *information* against additive attacks was previously considered in [CDF+08], who introduced AMD codes as a means of detecting such attacks. One of the motivating applications of AMD codes was that of converting any (passive-secure) linear secret sharing scheme into a similar scheme that offers error-detection in the presence of active tampering with the shares. Our application of AMD circuits to MPC is conceptually similar: we use AMD circuits to convert passive-secure MPC protocols with a certain "linear" structure into similar MPC protocols that offer security against active corruptions.

A concurrent and independent work of Ikarashi et al. [IKHC14] also makes the observation that in some natural passive-secure MPC protocols, any active attack on the protocol corresponds to an additive attack on the underlying circuit. This is then used to immunize such protocols against active attacks. The main result of [IKHC14] is a protocol with the same asymptotic complexity as the protocol of Section 5.8, which offers active security in the presence of an honest majority with a communication complexity of $O(n|C| + n^2)$ field elements.

The protocol of [IKHC14] is simpler, has better concrete complexity than our corresponding protocol, and slightly better security ($O(1/|\mathbb{F}|)$ vs. $O(|C|/|\mathbb{F}|)$). It relies on the fact that the additive attacks induced by active attacks on the passive-secure protocol from [DN07] (as well as the simpler BGW protocol [BGW88]) have a limited form: only outputs of multiplication gates and output gates are vulnerable to attacks, and the additive attack is consistent across all the wires connecting the output of a gate to the inputs of other gates. The protocol from [IKHC14] implicitly relies on a simple construction of AMD circuits that tolerate such restricted additive attacks. This construction does not protect against more general additive attacks. Protecting against general additive attacks seems necessary for securing other passive-secure protocols to which we apply our methodology, such as the semi-honest GMW protocol in the OLE hybrid model [GMW87, IPS09].

2 Preliminaries

2.1 Definitions

We begin by defining the notion of arithmetic circuits. We extend the standard notion of arithmetic circuits (Cf. [SY10]) by allowing randomness gates as well as gates the compute arbitrary functions.

Definition 2.1 (Arithmetic circuit)**.** *Let $\mathbb{F}$ be a finite field and let $\mathcal{G}$ be a set of functions $g_i : \mathbb{F}^{n_i} \to \mathbb{F}$. An arithmetic circuit C over the gate set $\mathcal{G}$ and a set of variables $X = \{x_1, \cdots, x_n\}$ is a directed acyclic graph whose vertices are called* gates *and whose edges are called* wires*. Every gate in C of in-degree 0 is either labeled by a variable from X and is referred to as an* input *gate or is labeled by the constant* one *and is referred to as* one *gate. All other gates are labeled by functions from $\mathcal{G}$. Every gate of out-degree 0 is called an* output *gate. We assume that the output gates are ordered. In some cases we also allow in-degree 0 gates labeled by* rand *and referred to as* randomness *gates. A circuit containing* rand *gates is called a* randomized *circuit and a circuit that does not contain* rand *gates is called a* deterministic *circuit.*

Unless stated otherwise, we shall assume that $\mathcal{G} = \{+, -, \times\}$. Gates labeled by $+$, $-$ and $\times$ are called addition, subtraction or multiplication gates respectively. We assume that the in-degree of addition, subtraction and multiplication gates is 2.

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. We denote by $|C|$ the number of gates in C. A one gate outputs the field element 1. Similarly, a randomness gate outputs a random element from $\mathbb{F}$. For an input $\mathbf{x} \in \mathbb{F}^n$ we denote by $C(\mathbf{x})$ the result of evaluation C on $\mathbf{x}$ if C is deterministic and the resulting distribution if C is randomized. For a pair of circuits $C_1 : \mathbb{F}^n \to \mathbb{F}^k$ and $C_2 : \mathbb{F}^k \to \mathbb{F}^t$ we define the circuit $C_2(C_1(\cdots))$ as the circuit obtained from connecting the outputs of C_1 to the inputs of C_2.

The degree of each gate in C, as well as the degree of C are defined as follows.

Definition 2.2 (Degree)**.** *We use $\mathsf{deg}(g)$ to denote the* degree *of gates defined as follows. If g is an input or* rand *gate then $\mathsf{deg}(g) = 1$. If g is a* one *gate then $\mathsf{deg}(g) = 0$. If $g = g_a + g_b$ or $g = g_a - g_b$ then $\mathsf{deg}(g) = \max(\mathsf{deg}(g_a), \mathsf{deg}(g_b))$. Finally, if $g = g_a \times g_b$ then $\mathsf{deg}(g) = \mathsf{deg}(g_a) + \mathsf{deg}(g_b)$. The degree of a circuit C, denoted by $\mathsf{deg}(C)$, is defined to be the maximal degree of all gates of C.*

Attack model. Intuitively, an additive attack A on a deterministic or randomized circuit C changes the computation by "blindly" adding a constant field element to every wire in C as well as to the outputs of C. Let $C : \mathbb{F}^n \to \mathbb{F}^k$ be a circuit. An *additive attack* A on C assigns an element of $\mathbb{F}$ to each of its internal wires as well as to each of the C's outputs. We denote by $A_{u,v}$ the attack A restricted to the wire (u, v). Similarly we denote by A_{out} the restriction of A to the outputs of C.

An additive attack A changes the computation performed by the circuit C as follows. For every wire (u, v) in C, the value $A_{u,v}$ is added to the output of u before it enters the inputs of v. Similarly, the value A_{out} is added to the outputs of C. For a circuit $C : \mathbb{F}^n \to \mathbb{F}^k$ and an additive attack A, we denote by $\widetilde{C} \leftarrow A(C)$ the circuit resulting from the additive attack A on C.

Notation. Let $\mathbb{F}$ be a finite field, we use $\mathbb{F}^*$ to denote both $\mathbb{F} \setminus \{0\}$ and the multiplicative group of $\mathbb{F}$. The meaning will be clear from the context. Let $\mathbf{u}, \mathbf{v} \in \mathbb{F}^n$. Denote by v_i the i'th coordinate of $\mathbf{v}$. Denote by $\mathbf{v} \cdot \mathbf{u}$ the inner product over $\mathbb{F}$ of $\mathbf{v}$ and $\mathbf{u}$. When appears inside a circuit construction we define $\mathbf{v} \cdot \mathbf{u}$ to be the circuit computing $\sum_{i=1}^{n} v_i \cdot u_i$ by multiplying the coordinates of $\mathbf{v}$ and $\mathbf{u}$ using n multiplication gates and then summing up the result using addition gates.

Notice that in the case of addition gates, any attack on their input wires has an equivalent attack on their output wires. Thus, we can assume without loss of generality that only the input wires of multiplication and output gates are attacked since the attack on inputs of addition gates can be pulled "downstream" through them until the inputs of multiplication or output gates.

Lemma 2.1. *Let C be a randomized arithmetic circuit and let A be an additive attack. Then there exists an additive attack A' that only attacks the inputs of multiplication and outputs of output gates of C such that for all x it holds that*

$$\widetilde{C}(x) \equiv \widetilde{C}'(x)$$

where $\widetilde{C} \leftarrow A(C)$ and $\widetilde{C}' \leftarrow A'(C)$.

Distributions. Denote by U_n the uniform distribution over $\mathbb{F}^n$ and denote by U^* the uniform distribution over $\mathbb{F}^*$. If two distributions D_1 and D_2 are the same we denote that by $D_1 \equiv D_2$. We will rely on the following standard claims.

Claim 2.1. *Let $\Delta_1, \Delta_2, \Delta_3$ be distributions such that $SD(\Delta_1, \Delta_2) = \epsilon$ and $SD(\Delta_2, \Delta_3) = \epsilon'$. Then, $SD(\Delta_1, \Delta_3) \leq \epsilon + \epsilon'$.*

Claim 2.2. *Let Δ_1, Δ_2 be two distributions such that $SD(\Delta_1, \Delta_2) \leq \epsilon$. Then, for any function f it holds that $SD(f(\Delta_1), f(\Delta_2)) \leq \epsilon$.*

2.2 AMD codes

In this section we study the goal of encoding information in a way that will allow us to pass it from one part of the circuit to the other without the adversary being able to alter the information in an undetectable way.

Intuitively, we would like to find a pair of arithmetic circuits $(\mathsf{Enc}, \mathsf{Dec})$ over some field $\mathbb{F}$ with the following two properties First, Dec can always recover x from the output of Enc invoked on input x. Second, any additive attack mounted on the outputs of Enc will be detected with some non-zero probability. Finally, we require that the size of $(\mathsf{Enc}, \mathsf{Dec})$ does not depend on the size of $\mathbb{F}$. Formally,

Definition 2.3. *An (n, k, ϵ)-AMD code is a pair of circuits $(\mathsf{Enc}, \mathsf{Dec})$ where $\mathsf{Enc} : \mathbb{F}^n \to \mathbb{F}^k$ is randomized and $\mathsf{Dec} : \mathbb{F}^k \to \mathbb{F}^{n+1}$ is deterministic such that the following properties hold:*

- Perfect completeness. *For all $\mathbf{x} \in \mathbb{F}^n$,*

$$\Pr[\mathsf{Dec}(\mathsf{Enc}(\mathbf{x})) = (0, \mathbf{x})] = 1.$$

- Additive robustness. *For any $\mathbf{a} \in \mathbb{F}^k$, $\mathbf{a} \neq 0$, and for any $\mathbf{x} \in \mathbb{F}^n$ it holds that*

$$\Pr[\mathsf{Dec}(\mathsf{Enc}(\mathbf{x}) + \mathbf{a}) \notin \mathsf{ERR}] \leq \epsilon.$$

We also define the notion of private *(n, k, ϵ)-AMD code that has the same completeness and security requirements as well as additional privacy requirement that for any $\mathbf{a} \in \mathbb{F}^k$, $\mathbf{a} \neq 0$, $y \in \mathbb{F}^* \times \mathbb{F}^k$, and for any $\mathbf{x}, \mathbf{x}' \in \mathbb{F}^n$ it holds that*

$$\Pr\left[\mathsf{Dec}(\mathsf{Enc}(\mathbf{x}) + \mathbf{a}) = y \,|\, \mathsf{Dec}(\mathsf{Enc}(\mathbf{x}) + \mathbf{a}) \in \mathsf{ERR}\right] = \Pr\left[\mathsf{Dec}(\mathsf{Enc}(\mathbf{x}') + \mathbf{a}) = y \,|\, \mathsf{Dec}(\mathsf{Enc}(\mathbf{x}') + \mathbf{a}) \in \mathsf{ERR}\right].$$

Notice that private AMD codes could be easily constructed from AMD codes by requiring the decoder to output random values in the case where a corruption is detected. Formally, consider Construction 2.1 below.

Construction 2.1. *Let $(\mathsf{Enc}, \mathsf{Dec})$ be a (n, k, ϵ)-AMD code. Consider the circuits $(\mathsf{Enc}', \mathsf{Dec}')$ that are defined as follows.*

- *The circuit Enc' on input $\mathbf{x}$ outputs $\mathsf{Enc}(\mathbf{x})$.*
- *The circuit Dec' on input $\mathbf{x}$ performs the following.*

1. *Compute* $(b, \mathbf{y}) \leftarrow \mathsf{Dec}(\mathbf{x})$.
2. *Output* $(0, \mathbf{y}) + b\mathbf{r}$ *where* $\mathbf{r}$ *is generated uniformly from* $\mathbb{F}^{n+1}$.

The following theorem can be easily verified.

Theorem 2.1. *For any pair of circuits* $(\mathsf{Enc}, \mathsf{Dec})$ *that are an* (n, k, ϵ)*-AMD code, the pair of circuits* $(\mathsf{Enc}', \mathsf{Dec}')$ *defined in Construction 2.1 are a private* (n, k, ϵ)*-AMD code.*

Asymptotically optimal constructions of such codes have been introduced by [DKRS06] and by [CDF+08].

Theorem 2.2 (Implicit in [CDF+08] Corollary 1)**.** *For any positive integers* n, σ *and field* $\mathbb{F}$ *there exists a pair of circuits* $(\mathsf{Enc}, \mathsf{Dec})$ *over* $\mathbb{F}$ *that are an* $(n, O(n+\sigma), \frac{1}{|\mathbb{F}|^\sigma})$*-AMD code. Moreover, the size of* Enc *and* Dec *is* $O(n+\sigma)$.

In fact, [CDF+08] consider a slightly weaker definition of AMD codes where is it guaranteed that $\Pr[\mathsf{Dec}(\mathsf{Enc}(\mathbf{x})+\mathbf{a}) \notin \mathsf{ERR} \cup \{(0, \mathbf{x})\}] \leq \epsilon$. However, their construction actually has the stronger security property of Definition 2.3. Combining Theorem 2.2 with the construction of private AMD codes outlined above we obtain the following:

Corollary 2.1. *For any positive integers* n, σ *there exists a pair of circuits* $(\mathsf{Enc}, \mathsf{Dec})$ *such that for any finite field* $\mathbb{F}$ *it holds that* $(\mathsf{Enc}, \mathsf{Dec})$ *are a private* $(n, O(n+\sigma), \frac{1}{|\mathbb{F}|^\sigma})$*-AMD code. Moreover, the size of* Enc *and* Dec *is* $O(n+\sigma)$.

3 AMD circuits over large finite fields

3.1 Simplifying the circuit model

Our constructions below assume that $\mathcal{G}$ is composed out of addition, subtraction and multiplication gates and do not consider circuit that contain one gates. It is possible to apply our constructions on a circuit C over a field $\mathbb{F}$ containing one gates as follows.

Construction 3.1 (Removing constants)**.** *Let* C *be a circuit containing* one *gates, construct the* ϵ*-secure implementation* $\widehat{C}$ *of* C *as follows.*

1. *Replace the* one *gates in* C *with an input gate* g *and obtain the circuit* C' *that outputs the output of* C *if the output of* g *is* 1 *and a random value otherwise (this can be implemented by generating a random value* r *and outputting* $z + (gr - r)$ *where* z *is the output of* C*).*
2. *Notice that* C' *does not contain* one *gates but assumes that the constant* 1 *is given in the input gate* g*. Let* $\widehat{C'}$ *we an* ϵ*-secure implementation of* C'.
3. *Construct the circuit* $\widehat{C}$ *obtained from* $\widehat{C'}$ *by connecting a* one *gate to* g.

The following lemma immediately follows from the fact that any additive attack on $\widehat{C}$ is equivalent to an additive attack on the inputs and outputs of C' and is therefore equivalent to an additive attack on the inputs and outputs of C and on the one gate used in C. Notice that any attack on the on one is caught by C' causing the outputs of $\widehat{C}$ to become random.

Lemma 3.1. *For any circuit* C *containing* one *gates, the circuit* $\widehat{C}$ *constructed in Construction 3.1 is an* $(\epsilon + \frac{1}{|\mathbb{F}|})$*-secure implementation of* C.

Remark 3.1. Our constructions for additive-attack correctness below also do not deal with circuits containing one gates. However, it is possible to apply them on circuits that contain one gates using an analog of Construction 3.1 and Lemma 3.1.

Unless stated otherwise, in the sequel we will assume that our constructions will be only applied on deterministic circuits that *do not* contain randomness gates. It is possible to apply our constructions on a circuit C over a field $\mathbb{F}$ containing randomness gates as follows.

Construction 3.2 (Removing randomness gates)**.** *Let C be a randomized circuit, construct the ϵ-secure implementation $\widehat{C}$ of C as follows.*

1. *First, replace every randomness gate g_i in C with an input gate g'_i and obtain the circuit C'. Let $\widehat{C'}$ be an ϵ-secure implementation of C'.*
2. *Construct the circuit $\widehat{C}$ obtained from $\widehat{C'}$ by replacing every g'_i created in the previous step with a randomness gate.*

The following lemma immediately follows from the fact that any additive attack on $\widehat{C}$ is equivalent to an additive attack on the inputs and outputs of C as well as on the randomness gates of C. Notice that any additive attack on the randomness gates of C does not effect the distribution of the obtained randomness.

Lemma 3.2. *For any randomized circuit C, the circuit $\widehat{C}$ constructed in Construction 3.2 is an ϵ-secure implementation of C.*

Remark 3.2. Our constructions for additive-attack correctness below also do not deal with circuits containing randomness gates. However, it is possible to apply them on circuits that contain randomness gates using an analog of Construction 3.2 and Lemma 3.2.

The security of guarantee of Theorem 1.1 directly relates to the size of the field $\mathbb{F}$ over which the circuit is defined. It is possible to amplify the security of Theorem 1.1 by computing C over a suitable extension field $\mathbb{H}$ of $\mathbb{F}$. Since the adversary is allowed to attack the inputs of a secure implementation of C, we cannot apply our constructions on C over $\mathbb{H}$ directly. Instead, we must modify C to a circuit C' over $\mathbb{H}$ that checks that the inputs to C are indeed elements of $\mathbb{F}$ and then apply our constructions on C'. Formally, consider the following theorem.

Theorem 3.1 (Amplifying additive-attack security)**.** *For any circuit $C : \mathbb{F}^n \to \mathbb{F}^k$ and $\sigma > 0$ there exists a randomized circuit $\widehat{C} : \mathbb{H}^n \to \mathbb{H}^k$ over some extension field $\mathbb{H}$ of $\mathbb{F}$ such that $\widehat{C}$ is a $O(2^{-\sigma})$-secure implementation of C.*

Proof. Let $\mathbb{H}$ be an extension field of $\mathbb{F}$ such that $|\mathbb{H}| \geq (|C| + n \log |\mathbb{F}|) \cdot 2^{\sigma}$. Notice that for any field element $x \in \mathbb{H}$ such that $x \neq 0$ it holds that $x^{|\mathbb{F}|-1} = 1$ if and only if $x \in \mathbb{F}$. Define a circuit $C'(\mathbf{x}) = C(\mathbf{x}) + \left(\sum_{i=1}^{n} s_i \cdot B\left(x_i^{|\mathbb{F}|-1}\right)\right) \cdot \mathbf{r}$ over $\mathbb{H}$ where $s_1, \cdots, s_n$ are random field elements generated using randomness gates inside C', $\mathbf{r}$ is a random vector over $\mathbb{H}^k$ generated using random gates inside C' and $B(x) = x(1-x)$ is a polynomial that vanishes only on 0 and 1. Notice that if the input does not belong to the field $\mathbb{F}$ then the output of C' is uniformly random and that $C'(\mathbf{x}) = C(\mathbf{x})$ for every input $\mathbf{x} \in \mathbb{F}^n$. In addition, by construction we have that $|C'| = O(|C| + n \log |\mathbb{F}|)$. Let $\widehat{C}$ be an $O(|C'|/|\mathbb{H}|)$-secure implementation of C'. Notice that since $|\mathbb{H}| \geq (|C| + n \log |\mathbb{F}|) \cdot 2^{\sigma}$ it holds that $\widehat{C}$ is an $O(2^{-\sigma})$-secure implementation of C as well. ■

3.2 Protecting low-degree circuits over large finite fields

In this section we construct ϵ-secure implementations for low-degree arithmetic circuits over large finite fields. The main idea behind the construction is as follows. We would like to ensure that any additive attack on the circuit will have one of two consequences: it will either cause the circuit to output a random output for all inputs, or it will be equivalent to a set of wire corruptions on the inputs and the

outputs of the circuit. To do so, we will encode the values in the circuit and compute over encoded values. The special property of the encoding is that every additive attack on the encoded values will cause the encoding to become invalid. We first present a simpler construction whose security holds when the wire values satisfy some local randomness property (Section 3.2.1). Later, we show how to eliminate this assumption by applying a general transformation to the circuit (Section 3.2.2). Finally, we combine the two together into a secure construction for low-degree circuits and arbitrary inputs (Section 3.2.3). We begin by presenting the security notion for specific input distributions.

Definition 3.1 (Additive-attack security with respect to a distribution)**.** *Let $\mathbb{F}$ be a finite field, $C : \mathbb{F}^n \to \mathbb{F}^k$ an arithmetic circuit, and I a distribution over $\mathbb{F}^n$. We say that a circuit $\widehat{C} : \mathbb{F}^n \to \mathbb{F}^k$ is an ϵ-secure implementation of C with respect to I if the following holds:*

- Completeness. *For all $\mathbf{x} \in \mathbb{F}^n$,*
$$\widehat{C}(\mathbf{x}) \equiv C(\mathbf{x}).$$
- Additive-attack security with respect to I. *For any additive attack A, there exists $\mathbf{a}^{\mathsf{in}} \in \mathbb{F}^n$ and a distribution $\mathcal{A}^{\mathsf{out}}$ over $\mathbb{F}^k$ such that*
$$SD\left(\widetilde{C}(I), C(I + \mathbf{a}^{\mathsf{in}}) + \mathcal{A}^{\mathsf{out}}\right) \leq \epsilon$$
where $\widetilde{C} \leftarrow A(\widehat{C})$.

3.2.1 Additive-attack security for locally-random distributions

We now present a secure construction for constant degree circuits and specific input distributions. Similarly to the approach of [BDOZ11, DPSZ12] for secure computation with preprocessing (and somewhat similarly to the MAC-based quantum MPC protocol of [BCG$^+$06]), our construction is based on a simple homomorphic MAC. However, in contrast to [BDOZ11, DPSZ12], we do not rely on any preprocessing phase to correctly produce MACed data, used to protect the circuit evaluation. Instead, our construction remains secure even when the sub-circuits computing and verifying the MAC tags are also subject to an additive attacks.

The main idea is to add for every wire in the circuit another wire carrying its MAC value. When two wires enter a gate the two MAC values corresponding to them will enter a special circuit that will produce the expected MAC value of the gate's result. Afterwards, the result of the gate and the corresponding MAC value are checked. The MAC used will have the property that if an input to a gate is attacked then the MAC value produced separately for this gate will not verify with the gate's result. As soon as this situation is detected a special abort flag will become non-zero causing the entire circuit to output a random value.

The construction will guarantee security as defined in Definition 3.1 with $\epsilon = O\left(\frac{d}{|\mathbb{F}|}\right)$, where d is the degree of the circuit it is applied on, under two assumptions.

1. The inputs of each gate are sufficiently random. In Section 3.2.2 we present a transformation that will randomize the inputs of each gate in the circuit.
2. The input to the circuit is taken from a specific input distribution. In Section 3.2.3 we present a construction for arbitrary inputs.

Since the security of the construction depends on the degree of the circuit, the construction is only useful for low-degree circuits. We start by defining what it means for inputs of each gate to be random and not to depend on the input to the circuit. Formally,

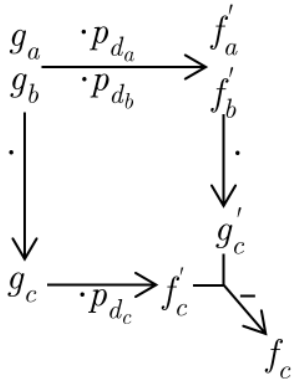


Figure 1: MAC computation for multiplication gates ($\cdot$ denotes field multiplication).

Definition 3.2. *Let $\mathbb{F}$ be a finite field, $C : \mathbb{F}^n \to \mathbb{F}^k$ a randomized arithmetic circuit, and I a distribution over $\mathbb{F}^n$. We say that C is locally ϵ-random with respect to I if for any $(y, z) \in \mathbb{F}^2$, and any pair of gates (g_1, g_2) whose outputs are the inputs to the same gate in C, it holds that the probability over $\mathbf{x} \leftarrow I$ and the internal randomness of C that the outputs of (g_1, g_2) in $C(\mathbf{x})$ are equal to (y, z) is at most ϵ.*

We now describe a construction that takes as input an ϵ-random circuit with respect to some class of input distributions and transforms it to a secure circuit with respect to the same class of distributions. The idea is as follows. For gate g_c with inputs g_a and g_b, the circuit will compute a MAC for g_c in two ways. The first way is by computing the gate's result and obtaining the MAC directly from the result. This MAC value is denoted in the construction below by f'_c. The second way is by homomorphically combining the input MACs f'_a and f'_b into a MAC for g_c. This MAC value is denoted below by g'_c. Finally, the circuit will verify that $f'_c = g'_c$. The guarantee of the MAC is that every additive attack is either harmless and will not affect the result, or it will be the case that $f'_c \neq g'_c$ with high probability. In the latter case, a special wire inside the circuit will become non-zero and will cause the entire circuit to output a random value. Intuitively, this guarantee is achieved by utilizing the fact that addition and multiplication do not commute.

See Figure 3.2.1 for the MAC for multiplication gates. Formally, consider Construction 3.3 below.

Construction 3.3. *Let $C : \mathbb{F}^n \to \mathbb{F}^k$ be a circuit. Let g_i, $1 \leq i \leq |C|$, denote the gates of C in some topological order. Define a circuit $\widehat{C}$ that on input $\mathbf{x}$ performs the following:*

1. *Compute $\mathbf{z} = C(\mathbf{x})$.*
2. *Generate a random field element $v \in \mathbb{F}$.*
3. *For $i = 1, \ldots, \mathsf{deg}(C)$ compute $p_i = v^i$ using multiplication gates.*
4. *For any gate g_i of degree d_i, compute the value $f'_i = g_i \cdot p_{d_i}$.*
5. *For each non-input gate g_c, $c = n+1, \ldots, |C|$, let g_a and g_b be its inputs and let d_a and d_b be their degrees. Compute the value g'_c as follows:*
   - *If g_c is a multiplication gate, let $g'_c = f'_a \cdot f'_b$.*
   - *If g_c is an addition gate: (1) if $d_a > d_b$ let $h'_c = p_{d_a - d_b} \cdot f'_b$ and $g'_c = f'_a + h'_c$, (2) if $d_a < d_b$ let $h'_c = p_{d_b - d_a} \cdot f'_a$ and $g'_c = h'_c + f'_b$, and (3) if $d_a = d_b$ let $g'_c = f'_a + f'_b$.*
   - *Similarly, if g_c is a subtraction gate: (1) if $d_a > d_b$ let $h'_c = p_{d_a - d_b} \cdot f'_b$ and $g'_c = f'_a - h'_c$, (2) if $d_a < d_b$ let $h'_c = p_{d_b - d_a} \cdot f'_a$ and $g'_c = h'_c - f'_b$, and (3) if $d_a = d_b$ let $g'_c = f'_a - f'_b$ if $d_a = d_b$.*
6. *For any non-input gate g_i, let $f_i = f'_i - g'_i$.*
7. *Let $f = \sum_i f_i r_i$ where r_i is a random field element.*
8. *Output $\mathbf{z} + f\mathbf{r}'$ where $\mathbf{r}'$ is a random vector from $\mathbb{F}^k$.*

Theorem 3.2. *Let $C : \mathbb{F}^n \to \mathbb{F}^k$ be a randomized arithmetic circuit of degree d which is locally ϵ-random with respect to a distribution I. Then the circuit $\widehat{C}$ obtained by applying Construction 3.3 to C is a $\left(|\mathbb{F}|\epsilon + \frac{d+1}{|\mathbb{F}|}\right)$-secure implementation of C with respect to I.*

Proof. The completeness property easily follows from the construction of $\widehat{C}$. To show the additive-attack security property for specific distributions, consider any additive attack A and let $\widetilde{C} \leftarrow A(\widehat{C})$.

In the following we will discuss only the corrupted circuit $\widetilde{C}$. Since all gates and values will be inside $\widetilde{C}$, for readability we will omit the tildes (e.g., g refers to $\widetilde{g}$).

We shall assume without loss of generality that the inputs to the circuit are given on special wires that are the inputs of input gates. Thus we redefine input gates as gates that have one input wire and compute the identity function of their inputs. We allow these special input wires to be attacked as well.

By Lemma 2.1 we can assume without loss of generality that A only attacks the inputs of multiplication gates inside $\widetilde{C}$. Hence we assume that for every gate g_c it holds that the corresponding wires (f'_c, f_c), (g'_c, f_c) are not attacked since they enter addition and subtraction gates. Moreover, for an addition or subtraction gate g_c with inputs g_a and g_b of degrees d_a and d_b we assume that the wires (g_a, g_c), (g_b, g_c) are not attacked, and also:

- The wires (f'_a, g'_c) and (h'_c, g'_c) are not attacked for the case of $d_a > d_b$.
- The wires (f'_b, g'_c) and (h'_c, g'_c) are not attacked for the case of $d_a < d_b$.
- The wires (f'_a, g'_c) and (f'_b, g'_c) are not attacked for the case of $d_a = d_b$.

For the case of an input gate g_c we can assume that A does not attack the wire (g_c, f'_c) since this wire attack is equivalent to attacking the input wire to g_c and modifying the wire attack on all other wires (except (g_c, f'_c)) by subtracting the wire attack on (g_c, f'_c) from it.

In the following we identify every value p_i, f_c, g'_c, f'_c (computed during the evaluation of $\widehat{C}$) with the corresponding (single) gate which computes this value. Let g_c be a gate inside C of degree d_c with inputs g_a and g_b of degrees d_a and d_b respectively. Notice that by the design of $\widehat{C}$ the subcircuits computing p_{d_a}, p_{d_b} and p_{d_c} inside $\widetilde{C}$ only have v is as input and only contain enough multiplication gates to compute some polynomial in v of degree d_a, d_b and d_c respectively even when subjected to additive attacks.

We define a gate g_c of degree d_c with inputs g_a and g_b of degrees d_a and d_b in $\widetilde{C}$ to be *problematic* if one of the following holds

- If $\widetilde{g}_c$ is a multiplication gate:

  Let $\widetilde{p}_{d_a}(v)$, $\widetilde{p}_{d_b}(v)$ and $\widetilde{p}_{d_c}(v)$ be the formal polynomials computed by the circuits computing p_{d_a}, p_{d_b} and p_{d_c} respectively. Then g_c is problematic if at least on the following holds:

  1. $A_{g_a,g_c} \neq 0$ or $A_{g_b,g_c} \neq 0$.
  2. $A_{f'_a,g'_c} \neq 0$ or $A_{f'_b,g'_c} \neq 0$.
  3. $\widetilde{p}_{d_c}(v) + A_{p_{d_c},f'_c} \neq (\widetilde{p}_{d_a}(v) + A_{p_{d_a},f'_a}) \cdot (\widetilde{p}_{d_b}(v) + A_{p_{d_b},f'_b})$.
  4. $A_{g_c,f'_c} \neq 0$.
  5. $A_{f_c,f} \neq 0$.

- If g_c is an addition or subtraction gate:

  Let $\widetilde{p}_a(v)$, $\widetilde{p}_b(v)$ and $\widetilde{p}_c(v)$ be the formal polynomials computed by the circuits computing p_{d_a}, p_{d_b} and p_{d_c} respectively. Also, if $d_a \neq d_b$ let $\widetilde{p}_{|d_a-d_b|}(v)$ be the formal polynomial computed by the circuit computing $p_{|d_a-d_b|}$. Then g_c is problematic if at least on the following holds:

  1. $d_a > d_b$, and either $(\widetilde{p}_{|d_a-d_b|}(v) + A_{p_{|d_a-d_b|},h'_c}) \cdot (\widetilde{p}_{d_b}(v) + A_{p_{d_b},f'_b}) \neq \widetilde{p}_{d_c}(v) + A_{p_{d_c},f'_c}$ or $\widetilde{p}_{d_a}(v) + A_{p_{d_a},f'_a} \neq \widetilde{p}_{d_c}(v) + A_{p_{d_c},f'_c}$ or $A_{g'_b,h'_c} \neq 0$.

2. $d_a < d_b$, and either $(\widetilde{p}_{|d_a-d_b|}(v) + A_{p_{|d_a-d_b|},h'_c}) \cdot (\widetilde{p}_{d_a}(v) + A_{p_{d_a},f'_a}) \neq \widetilde{p}_{d_c}(v) + A_{p_{d_c},f'_c}$ or $\widetilde{p}_{d_b}(v) + A_{p_{d_b},f'_b} \neq \widetilde{p}_{d_c}(v) + A_{p_{d_c},f'_c}$ or $A_{g'_a,h'_c} \neq 0$.
3. $d_a = d_b$, and either $\widetilde{p}_{d_a}(v) + A_{p_{d_a},f'_a} \neq \widetilde{p}_{d_c}(v) + A_{p_{d_c},f'_c}$ or $\widetilde{p}_{d_b}(v) + A_{p_{d_b},f'_b} \neq \widetilde{p}_{d_c}(v) + A_{p_{d_c},f'_c}$.
4. $A_{f_c,f} \neq 0$.
5. $A_{g_c,f'_c} \neq 0$ for the case where g_c is an addition gate or $A_{g_c,f'_c} \neq 0$ for the case where g_c is a subtraction gate.

We prove the following inner lemma.

Lemma 3.3. *If $\widetilde{C}$ does not contain any problematic gates then every gate originating from C is computed correctly (i.e. for every multiplication, addition or subtraction gate g_c with inputs g_a and g_b it holds that $g_c = g_a \cdot g_b$, $g_c = g_a + g_b$ or $g_c = g_a - g_b$ if the gate is a multiplication, addition or subtraction gate respectively). In addition, it holds that $f'_c = g_c \cdot (p_{d_c} + A_{p_{d_c},f'_c})$ for every such gate g_c, and that $f'_i = g'_i$ and $f_c = 0$ for non-input gates.*

Proof. The proof is by induction on the circuit structure.

Basis. For the case of input gates the claim holds since by the assumption we have that for any input gate g_c it holds that $A_{g_c,f'_c} = 0$.

Induction step. Let g_c be a non problematic gate of degree d_c and let g_a and g_b be inputs to g_c of degrees d_a and d_b such that both g_a and g_b are not problematic. Thus we have that our claim holds for g_a and g_b. We split the proof of the claim into three cases

- g_c is a multiplication gate. The circuits computing g_c actually compute the values

$$g_c = (g_a + A_{g_a,g_c})(g_b + A_{g_b,g_c}) = g_a \cdot g_b.$$

The last transition holds from the fact that g_c is not problematic. Next, the circuit computing g'_c actually computes

$$\begin{aligned} g'_c &= (f'_a + A_{f'_a,g'_c})(f'_b + A_{f'_b,g'_c}) = f'_a \cdot f'_b = g'_a \cdot g'_b = g_a g_b (\widetilde{p}_{d_a} + A_{p_{d_a},f'_a})(\widetilde{p}_{d_b} + A_{p_{d_b},f'_b}) \\ &= g_a g_b (\widetilde{p}_{d_c} + A_{p_{d_c},f'_c}) = g_c(\widetilde{p}_{d_c} + A_{p_{d_c},f'_c}) = f'_c. \end{aligned}$$

The second and fifth transitions hold since g_c is not problematic, and the third and forth transitions follow from the induction hypothesis. Finally, the circuit computing f_c actually computes

$$f_c = g_c(\widetilde{p}_{d_c} + A_{p_{d_c},f'_c}) - g'_c = g_c(\widetilde{p}_{d_c} + A_{p_{d_c},f'_c}) - g_c(\widetilde{p}_{d_c} + A_{p_{d_c},f'_c}) = 0.$$

- g_c is an addition gate, $d_a > d_b$. Since g_c is not problematic we have that $A_{g_c,f'_c} = 0$ Thus, the circuits computing g_c actually compute the values

$$g_c = g_a + A_{g_a,g_c} + g_b + A_{g_b,g_c} = g_a + g_b.$$

Next, the circuit computing g'_c actually computes

$$\begin{aligned} g'_c &= g'_a + h'_c \\ &= g'_a + (\widetilde{p}_{d_a-d_b} + A_{p_{d_a-d_b},h'_c}) \cdot (g'_b + A_{g'_b,h'_c}) \\ &= g_a(\widetilde{p}_a + A_{p_{d_a},f'_a}) + (\widetilde{p}_{d_a-d_b} + A_{p_{d_a-d_b},h'_c}) \cdot g'_b \\ &= g_a(\widetilde{p}_a + A_{p_{d_a},f'_a}) + (\widetilde{p}_{d_a-d_b} + A_{p_{d_a-d_b},h'_c}) \cdot g_b \cdot (p_{d_b} + A_{p_{d_b},f'_b}) \\ &= g_a(\widetilde{p}_c + A_{p_{d_c},f'_c}) + g_b \cdot (\widetilde{p}_c + A_{p_{d_c},f'_c}) \\ &= (g_a + g_b)(\widetilde{p}_c + A_{p_{d_c},f'_c}) \\ &= f'_c \end{aligned}$$

Finally, the circuit computing f_c actually computes

$$f_c = g_c \cdot \widetilde{p}_{d_c} - g'_c = (g_a + g_b)\widetilde{p}_{d_c} - (g_a + g_b)\widetilde{p}_{d_c} = 0.$$

The first transition follow from the assumption that A does not attack both inputs to f_c.

- The case where $d_a \leq d_b$ and the case where g_c is a subtraction gadget are treated similarly and the proof is omitted.

This concludes the proof of Lemma 3.3. ■

We split the remainder of the proof of Theorem 3.2 into two cases.

There are no problematic gates. We will now prove that if $\widetilde{C}$ does not contain any problematic gates then there exists $\mathbf{a}^{\mathsf{in}} \in \mathbb{F}^n$ and a distribution $\mathcal{A}^{\mathsf{out}}$ over $\mathbb{F}^k$ such that

$$SD\left(\widetilde{C}(I), C\left(I + \mathbf{a}^{\mathsf{in}}\right) + \mathcal{A}^{\mathsf{out}}\right) = 0.$$

Indeed, note that by the above claim and the fact that all the gadgets are not problematic it holds that $f = \sum_c (f_c + A_{f_c,f}) r_c = 0$. In addition, notice that the claim above means that all the gates in the circuit where computed correctly. Let $\mathbf{a}^{\mathsf{in}}$ be A restricted to the wires carrying $\mathbf{x}$ to the input gates of $\widehat{C}$ and let $\mathbf{a}^{\mathsf{out}}$ be A restricted to the output gates of $\widehat{C}$. Let $\mathcal{A}^{\mathsf{out}}$ be the distribution over $\mathbb{F}^k$ defined as follows: For all $1 \leq t \leq k$ if the wire carrying f into the circuit outputting $z_t + fr'_t$ is attacked then $\mathcal{A}^{\mathsf{out}}_t = U_1$. Otherwise, $\mathcal{A}^{\mathsf{out}}_t = \mathbf{a}^{\mathsf{out}}_t$. We have that

$$SD\left(\widetilde{C}(I), C\left(I + \mathbf{a}^{\mathsf{in}}\right) + \mathcal{A}^{\mathsf{out}}\right) \leq 0.$$

There are problematic gates. Recall that the gates are sorted in topological order. Let g_c of degree d_c be the first problematic gate in $\widetilde{C}$. Let g_a and g_b be the inputs to g_c and let d_a and d_b be the degrees of g_a and g_b respectively. We split the proof into three cases.

- Suppose the minimal problematic gate is a multiplication gate. Since g_c is the minimal problematic gate the MAC values of g_a and g_b are computed correctly. Thus, it holds that $g_a \cdot (\widetilde{p}_{d_a} + A_{p_{d_a},f'_a}) = g'_a$ and that $g_b \cdot (\widetilde{p}_{d_b} + A_{p_{d_b},f'_b}) = g'_b$. Notice that A causes f_c to compute the following expression

$$\begin{aligned} f_c \;=\; & ((g_a + A_{g_a,g_c})(g_b + A_{g_b,g_c}) + A_{g_c,f'_c})(\widetilde{p}_c + A_{p_{d_c},f'_c}) - \\ & (g_a \cdot (\widetilde{p}_{d_a} + A_{p_{d_a},f'_a}) + A_{g'_a,g'_c})(g_b \cdot (\widetilde{p}_{d_b} + A_{p_{d_b},f'_b}) + A_{g'_b,g'_c}) + A_{f_c,f}. \end{aligned} \tag{1}$$

We view f_c as a polynomial in g_a, g_b and v.

If $(\widetilde{p}_{d_a} + A_{p_{d_a},f'_a}) \cdot (\widetilde{p}_{d_b} + A_{p_{d_b},f'_b}) \neq (\widetilde{p}_{d_c} + A_{p_{d_c},f'_c})$ then the coefficient $(\widetilde{p}_{d_c} + A_{p_{d_c},f'_c}) - (\widetilde{p}_{d_a} + A_{p_{d_a},f'_a}) \cdot (\widetilde{p}_{d_b} + A_{p_{d_b},f'_b})$ of the term $g_a \cdot g_b$ inside f_c is not always zero. Thus, by the Schwartz-Zippel lemma we have that

$$SD(\widetilde{C}(I), U_k) \leq |\mathbb{F}|\epsilon + \frac{d+1}{|\mathbb{F}|}.$$

Otherwise, it holds that $(\widetilde{p}_{d_a} + A_{p_{d_a},f'_a}) \cdot (\widetilde{p}_{d_b} + A_{p_{d_b},f'_b}) = (\widetilde{p}_{d_c} + A_{p_{d_c},f'_c})$. Since g_c is problematic, it must be the case that one of the following A_{g_a,g_c}, A_{g_b,g_c}, A_{g_c,f'_c}, $A_{g'_a,g'_c}$, $A_{g'_b,g'_c}$, and $A_{f_c,f}$ is not zero. Rewriting Equation 1 above we obtain

$$\begin{aligned} f_c \;=\; & A_{g_a,g_c} g_b (\widetilde{p}_{d_c} + A_{p_{d_c},f'_c}) + A_{g_b,g_c} g_a (\widetilde{p}_{d_c} + A_{p_{d_c},f'_c}) + (A_{g_a,g_c} A_{g_b,g_c} + A_{g_c,f'_c})(\widetilde{p}_{d_c} + A_{p_{d_c},f'_c}) + \\ & A_{g'_b,g'_c} g_a (\widetilde{p}_{d_a} + A_{p_{d_a},f'_a}) + A_{g'_a,g'_c} g_b (\widetilde{p}_{d_b} + A_{p_{d_b},f'_b}) + A_{g'_a,g'_c} A_{g'_b,g'_c} + A_{f_c,f}. \end{aligned} \tag{2}$$

Notice that $\widetilde{p}_{d_c} + A_{p_{d_c},f'_c}$ is a polynomial in v of degree d_c at most. If $A_{g_a,g_c} \neq 0$ the term $\widetilde{g}_b(\widetilde{p}_{d_c} + A_{p_{d_c},f'_c})$ has a non zero coefficient A_{g_a,g_c}. Thus, by the Schwartz-Zippel lemma we have that

$$SD(\widetilde{C}(I), U_k) \leq |\mathbb{F}|\epsilon + \frac{d+1}{|\mathbb{F}|}.$$

Similarly, using the terms $A_{g_b,g_c}g_a(\widetilde{p}_{d_c} + A_{p_{d_c},f'_c})$, $A_{g_c,f'_c}(\widetilde{p}_{d_c} + A_{p_{d_c},f'_c})$, $A_{g'_b,g'_c}g_a \cdot (\widetilde{p}_{d_a} + A_{p_{d_a},f'_a})$, $A_{g'_a,g'_c}g_b \cdot (\widetilde{p}_{d_b} + A_{p_{d_b},f'_b})$ and $A_{f_c,f}$ it can be shown that if A_{g_b,g_c}, A_{g_c,f'_c}, $A_{g'_a,g'_c}$, $A_{g'_b,g'_c}$ and $A_{f_c,f}$ are not zero then by the Schwartz-Zippel lemma we have that

$$SD(\widetilde{C}(I), U_k) \leq |\mathbb{F}|\epsilon + \frac{d+1}{|\mathbb{F}|}.$$

- Suppose the minimal problematic gate is an addition gate and that $d_a > d_b$ (without loss of generality). Since g_c is the minimal problematic gate the MAC values of g_a and g_b are computed correctly. Thus, it holds that $g_a \cdot (\widetilde{p}_{d_a} + A_{p_{d_a},f'_a}) = g'_a$ and that $g_b \cdot (\widetilde{p}_{d_b} + A_{p_{d_b},f'_b}) = g'_b$. Notice that A causes f_c to compute the following expression

$$\begin{aligned} f_c \;=\; & (g_a + g_b + A_{g_c,f'_c})(\widetilde{p}_{d_c} + A_{p_{d_c},f'_c}) - \\ & g_a \cdot (\widetilde{p}_{d_a} + A_{p_{d_a},f'_a}) - (\widetilde{p}_{|d_a-d_b|}(v) + A_{p_{|d_a-d_b|},h'_c}) \cdot (\widetilde{p}_{d_b}(v) + A_{p_{d_b},f'_b}) \cdot g_b - \\ & (\widetilde{p}_{|d_a-d_b|}(v) + A_{p_{|d_a-d_b|},h'_c}) \cdot A_{g'_b,h'_c}. \end{aligned} \tag{3}$$

  Again, we view f_c as a polynomial in g_a, g_b and v. If $\widetilde{p}_{d_a}(v) + A_{p_{d_a},f'_a} \neq \widetilde{p}_{d_c}(v) + A_{p_{d_c},f'_c}$ then the coefficients of g_a inside f_c are not always zero. Similarly, if $(\widetilde{p}_{|d_a-d_b|}(v) + A_{p_{|d_a-d_b|},h'_c})(\widetilde{p}_{d_b}(v) + A_{p_{d_b},f'_b}) \neq \widetilde{p}_{d_c}(v) + A_{p_{d_c},f'_c}$ then the coefficients of g_b are not always zero (again inside f_c). In both cases, by the Schwartz-Zippel lemma we have that

$$SD(\widetilde{C}(I), U_k) \leq |\mathbb{F}|\epsilon + \frac{d+1}{|\mathbb{F}|}.$$

  Otherwise, if $A_{g_c,f'_c} \neq 0$ then the non-zero polynomial $\widetilde{p}_{d_c}(v) + A_{p_{d_c},f'_c}$ is multiplied by a non-zero constant inside f_c. Similarly, if $A_{g'_b,h'_c} \neq 0$ then the non-zero polynomial $(\widetilde{p}_{|d_a-d_b|}(v) + A_{p_{|d_a-d_b|},h'_c})$ is multiplied by a non-zero constant inside f_c In both cases, by the Schwartz-Zippel lemma we have that

$$SD(\widetilde{C}(I), U_k) \leq \frac{d+1}{|\mathbb{F}|}.$$

  Also notice that if $A_{f,f_c} \neq 0$ then again by the Schwartz-Zippel lemma we have that

$$SD(\widetilde{C}(I), U_k) \leq \frac{1}{|\mathbb{F}|}.$$

- The case where the minimal problematic gadget is a subtraction gadget is treated similarly and the proof is omitted.

This concludes the proof of Theorem 3.2 above.

■

3.2.2 Obtaining locally-random circuits

In this section we present a general transformation mapping any arithmetic circuit C into a locally random circuit C' whose output encodes the output of C. Similar transformations were previously used for the purpose of protecting circuits against leakage [ISW03, IKO+11]. Here we show that a natural

generalization of a transformation from [IKO+11] to the arithmetic setting and show that it satisfies the required local randomness property, namely that the pair of inputs to each gate in C' have almost full entropy. Similarly to [IKO+11], each wire a inside C (including input and output wires) will be split into two wires, one masked by r_1 and the other masked by r_2. Each gate c of C with inputs a, b will be replaced by a gadget that maps $(a + r_1, a + r_2, b + r_1, b + r_2)$ to $(c + r_1, c + r_2)$. The gadget has the property that the inputs of every internal gate are almost completely random (assuming that r_1 and r_2 are random). The two random field elements r_1, r_2 will be reused for the whole circuit.

Construction 3.4. *The gadget* add *is the circuit that, on input* $(r_1, r_2, v_1, v_2, v_3, v_4)$ *performs the following: (the circuit will be always used where* $v_1 = a + r_1$, $v_2 = a + r_2$, $v_3 = b + r_1$, $v_4 = b + r_2$*)*

1. *Compute* $v_5 = v_1 + v_4$ *(note that* $v_5 = a + b + r_1 + r_2$*).*
2. *Compute* $v_6 = v_5 - r_2$ *(note that* $v_6 = a + b + r_1$*).*
3. *Compute* $v_7 = v_5 - r_1$ *(note that* $v_7 = a + b + r_2$*).*
4. *Output* (v_6, v_7)*.*

Similarly, the gadget sub *is the circuit that, on input* $(r_1, r_2, v_1, v_2, v_3, v_4)$ *performs the following: (the circuit will be always used where* $v_1 = a + r_1$, $v_2 = a + r_2$, $v_3 = b + r_1$, $v_4 = b + r_2$*)*

1. *Compute* $v_5 = v_1 - v_4$ *(note that* $v_5 = a - b + r_1 - r_2$*).*
2. *Compute* $v_6 = v_5 + r_2$ *(note that* $v_6 = a - b + r_1$*).*
3. *Compute* $v_7 = v_5 + r_2$ *(note that* $v_7 = a - b + r_1 + r_2$*).*
4. *Compute* $v_8 = v_7 - r_1$ *(note that* $v_7 = a - b + r_2$*).*
5. *Output* (v_6, v_8)*.*

The following obvious lemma show that the distribution of the inputs of every gate is indeed uniform.

Lemma 3.4. *For any values* $a, b \in \mathbb{F}$*, for any two wires* (w_1, w_2) *that are the inputs to the same gate in the circuits* add, sub *defined in Construction 3.4 and for any* $x, y \in \mathbb{F}$ *we have that*

$$\Pr_{r_1, r_2}[(w_1, w_2) = (x, y)] \leq \frac{1}{|\mathbb{F}|^2}.$$

Similarly, we define the multiplication gadget mult as follows:

Construction 3.5. *The gadget* mult *is the circuit that, on input* $(r_1, r_2, v_1, v_2, v_3, v_4)$ *performs the following: (the circuit will be always used where* $v_1 = a + r_1$, $v_2 = a + r_2$, $v_3 = b + r_1$, $v_4 = b + r_2$*):*

1. *Compute* $v_5 = v_1 v_4$ *(note that* $v_5 = ab + r_1 b + r_2 a + r_1 r_2$*).*
2. *Compute* $v_6 = v_1 r_2$ *(note that* $v_6 = a r_2 + r_1 r_2$*).*
3. *Compute* $v_7 = v_4 r_1$ *(note that* $v_7 = b r_1 + r_1 r_2$*).*
4. *Compute* $v_8 = v_5 + r_2$ *(note that* $v_8 = ab + r_1 b + r_2 a + r_1 r_2 + r_2$*).*
5. *Compute* $v_9 = v_8 - v_6$ *(note that* $v_9 = ab + r_1 b + r_2$*).*
6. *Compute* $v_{10} = v_9 - v_7$ *(note that* $v_{10} = ab + r_2 - r_1 r_2$*).*
7. *Compute* $v_{11} = r_1 r_2$

8. *Compute* $v_{12} = v_{10} + v_{11}$ *(note that* $v_{12} = ab + r_2$*).*

9. *Similarly, compute* $v_{13} = ab + r_1$

10. *Output* (v_{12}, v_{13})

Lemma 3.5. *For any values* $a, b \in \mathbb{F}$*, for any two wires* (w_1, w_2) *that are the inputs to the same gate in the circuit* mult *defined in Construction 3.5 above and for any* $x, y \in \mathbb{F}$ *we have that*

$$\Pr_{r_1,r_2}[(w_1, w_2) = (x, y)] \leq \frac{2}{|\mathbb{F}|^2}.$$

Using constructions 3.4 and 3.5 we can now randomize general circuits as follows. We start from a circuit C that contains only addition, subtraction and multiplication gates. We replace all multiplication, addition and subtraction gates with the gadgets from Constructions 3.5 and 3.4 respectively and obtain C'. Notice that both gadgets assume that their inputs are encoded is a specific way. That is, every input wire x is split into two wires $x + r_1$ and $x + r_2$. We thus add additional gates to C' that generate two new random values r_1, r_2 and encode every input x_i of C to $(x_i + r_1, x_i + r_2)$. We also require that C' will output r_1 and the first element of each wire pair. This will allow us to define a decoder circuit Dec in order to decode the outputs. Formally, consider Construction 3.6 below.

Construction 3.6. *Let* $C : \mathbb{F}^n \to \mathbb{F}^k$ *be a circuit. Consider the circuits* (C', Dec) *that are defined as follows.*

- *The circuit* C' *is constructed as follows.*
  1. C' *on input* $\mathbf{x}$ *generates random field elements* $r_1, r_2 \in \mathbb{F}$ *for every* i *computes* $x'_i = (x_i + r_1, x_i + r_2)$.
  2. *Every gate of* C *is replaced in* C' *by the corresponding gadget as described above.*
  3. *Let* $y_1, \ldots, y_k$ *be the first elements from wire pairs corresponding to the output.* C' *will output* $(r_1, y_1, \ldots, y_k)$.
- *The circuit* Dec *on input* $(r_1, y_1, \ldots, y_k)$ *outputs* $(y_1 - r_1, \ldots, y_k - r_1)$.

We now claim that the circuits resulting from Construction 3.6 are an $O\left(\frac{1}{|\mathbb{F}|^2}\right)$-random implementation of C with respect to all input distributions in which every input element is (individually) uniform. Formally,

Theorem 3.3. *For any circuit* $C : \mathbb{F}^n \to \mathbb{F}^k$*, the circuits* (C', Dec) *resulting from applying Construction 3.6 to* C *have the following properties:*

- Completeness. *For any* $\mathbf{x} \in \mathbb{F}^n$ *it holds that* $C(\mathbf{x}) = \mathsf{Dec}(C'(\mathbf{x}))$.
- Randomization. *The circuit* C' *is a* $\frac{2}{|\mathbb{F}|^2}$*-random circuit with respect to every input distribution* I *in which each entry* I_j *is distributed uniformly over* $\mathbb{F}$.
- Complexity. *The size of* C' *is* $O(|C|)$.

Proof. The completeness and complexity properties easily follow from the construction. As for the randomization property, we need the prove that the inputs of every gate inside C' are random. Indeed, notice that every the inputs of every gate that is part of the gadgets add, sub and mult are $\frac{2}{|\mathbb{F}|^2}$-random. Next, for every addition gate created during Step 1 of Construction 3.6, one of its inputs is a wire from circuit's input that looks uniform in I and the other is an output of a randomness gate. ■

3.2.3 Additive-attack security for arbitrary inputs

We now present our construction for additive-attack security of low-degree circuits for arbitrary inputs. We use a simple randomized input encoding to ensure that the input distribution I satisfies the required local randomness property.

Construction 3.7. *Let $C : \mathbb{F}^n \to \mathbb{F}^k$ be a circuit. We define the circuit $C_{\mathsf{AUG}} : \mathbb{F}^{n+1} \to \mathbb{F}^k$ that on input $x_0, \ldots, x_n$ outputs $C(x_1 - x_0, \ldots, x_n - x_0)$.*

We are now ready to present the main construction to secure low-degree circuits over arbitrary inputs.

Construction 3.8. *Let $C : \mathbb{F}^n \to \mathbb{F}^k$ be a circuit. We construct $\widehat{C}$ from C via the following transformations.*

- *Construct C_{AUG} from C using Construction 3.7 (to unmask the inputs of C).*
- *Construct (C', Dec) from C_{AUG} using Construction 3.6 (to randomize all wires inside C_{AUG}).*
- *Construct $\widehat{C}'$ from C' using Construction 3.3 (to additively secure C').*

Consider the circuit $\widehat{C}$ that on input $\mathbf{x}$ preforms the following.

- *Generate a random field element $r \in \mathbb{F}$.*
- *Compute $\mathbf{x}' = (r, x_1 + r, \ldots, x_n + r)$.*
- *Compute $\mathbf{y} = \widehat{C}'(\mathbf{x}')$.*
- *Output $\mathsf{Dec}(\mathbf{y})$.*

We claim that Construction 3.8 above transforms any low-degree circuit C to an additively secure circuit $\widehat{C}$.

Theorem 3.4. *For any arithmetic circuit C of degree d the circuit $\widehat{C}$ obtained by applying Construction 3.8 to C is an $O\left(\frac{d}{|\mathbb{F}|}\right)$-secure implementation of C.*

Proof. The completeness property easily follows from the construction of $\widehat{C}$. To show the additive-attack security property, consider any additive attack A. Since the subcircuits computing $\mathbf{x}'$ and $\mathsf{Dec}(\mathbf{y})$ only contain addition and subtraction gates we can assume without loss of generality that A only attacks the circuit $\widehat{C}_{\mathsf{AUG}}$ inside $\widehat{C}$. For any $\mathbf{x} \in \mathbb{F}^n$ define the distribution $\mathcal{X} = (r, x_1 + r, \ldots, x_n + r)$ where r is a random element from $\mathbb{F}$. By Theorem 3.3 we have that for any $\mathbf{x}$ it holds that C' is a $\frac{2}{|\mathbb{F}^2|}$-random circuit with respect to the distribution $\mathcal{X}$. Thus, by Theorem 3.2 we have that there exists $\mathbf{a}'^{\mathsf{in}} \in \mathbb{F}^{n+1}$ and a distribution $\mathcal{A}'^{\mathsf{out}}$ over $\mathbb{F}^{k+1}$ such that for any $\mathbf{x} \in \mathbb{F}^n$

$$SD\left(\widetilde{C}_{\mathsf{AUG}}(\mathcal{X}), C_{\mathsf{AUG}}\left(\mathcal{X} + \mathbf{a}'^{\mathsf{in}}\right) + \mathcal{A}'^{\mathsf{out}}\right) \leq O\left(\frac{d}{|\mathbb{F}|}\right)$$

where $\widetilde{C}_{\mathsf{AUG}} \leftarrow A(\widehat{C}_{\mathsf{AUG}})$. Define the distribution $\mathcal{A}^{\mathsf{out}} = \mathsf{Dec}(\mathcal{A}'^{\mathsf{out}})$. By claim 2.2 we have that

$$SD\left(\mathsf{Dec}(\widetilde{C}_{\mathsf{AUG}}(\mathcal{X})), \mathsf{Dec}\left(C_{\mathsf{AUG}}\left(\mathcal{X} + \mathbf{a}'^{\mathsf{in}}\right)\right) + \mathcal{A}^{\mathsf{out}}\right) \leq O\left(\frac{d}{|\mathbb{F}|}\right).$$

In addition, define $\mathbf{a}^{\mathsf{in}} = (a_1'^{\mathsf{in}} - a_0'^{\mathsf{in}}, \ldots, a_n'^{\mathsf{in}} - a_0'^{\mathsf{in}})$. Thus we have that for all $\mathbf{x} \in \mathbb{F}^n$

$$SD\left(\widetilde{C}(\mathbf{x}), C\left(\mathbf{x} + \mathbf{a}^{\mathsf{in}}\right) + \mathcal{A}^{\mathsf{out}}\right) \leq O\left(\frac{d}{|\mathbb{F}|}\right).$$

■

3.2.4 A counterexample for small fields

Construction 3.8 above assumes the existence of a way to secure constant degree circuit with respect to a specific class of input distributions. This goal is achieved by Construction 3.3 above. Unfortunately, over the binary field, Construction 3.3 fails to achieve any meaningful security. The following is an attack that will allow one to flip arbitrary wires that are the inputs to a multiplication gadget even when the inputs of the multiplication gate corresponding to it are completely random. Let C be a ϵ-locally-random circuit over the binary field with respect to some input distribution I. By construction 3.3 for every multiplication gate g_c inside C with inputs g_a and g_b we have that $\widehat{C}$ adds another multiplication gate g'_c with inputs g'_a and g'_b. Notice that every multiplication gate in the circuit is verified by computing $f_c = g_c \cdot p_{d_c} - g'_c = g_c \cdot p_{d_c} - g'_a \cdot g'_b$ where d_c is the degree of g_c. Theorem 3.2 proves that it is impossible to modify g_a and g_b without f_c becoming non-zero with high probability.

Unfortunately, this is true only for large fields. Consider an attack that adds the constant 1 to g_a and g'_a. Notice that

$$f_c = g_c \cdot p_{d_c} - (g'_a + 1) \cdot g'_b = (g_a + 1)g_b v^{d_c} - g_a v^{d_a} \cdot g_b v^{d_b} + v^{d_b} g_b = (g_a + 1)g_b v - v \cdot g_a \cdot g_b + v g_b = 0$$

which passes the check of $f_c = 0$. The main reason why this attack is successful its that over the binary field it holds that $v^{d_c} = v^{d_a} = v^{d_b} = v$ for all $d_a, d_b, d_c \in \mathbb{N}$. This can also be seen in the diagram presented in Figure 3.2.1 above. Notice that for the case of the binary field, multiplying g_a and g_b by p_{d_a} and p_{d_b} obtaining f'_a and f'_b respectively and then multiplying both f'_a and f'_b is always the same as multiplying g_a by g_b and by p_{d_c} since it is the case that $p_{d_c} = p_{d_a} = p_{d_b} = v$. Thus, we have managed to modify the value of a wire inside $\widehat{C}$ even when the inputs of each multiplication gate are completely random.

3.3 Protecting arbitrary circuits over large finite fields

In Section 3.2 we presented a transformation that meets the goal of additive-attack security (see Definition 1.1) for low-degree circuits. However, the field size required for obtaining security grows linearly with the degree of the circuit. Thus, in some cases, the field size must grow exponentially with the circuit size. In this section, we would like to present a transformation where the field size can be much smaller. We first present a construction using small tamper-proof components and later eliminate these components.

3.3.1 A solution using tamper-proof components

As an intermediate step, we show a construction that assumes tamper-proof components (which will be subsequently replaced by gadgets built of regular gates, see Section 3.3.2). Suppose we are given tamper-proof components G_{add}, G_{sub} and G_{mul}, that receive a pair of AMD-encoded inputs for a gate g, decode the inputs, compute the output of g, and finally produce a fresh AMD encoding of this output.

Construction 3.9. *Let* $(\mathsf{Enc}, \mathsf{Dec})$ *be a private* $(1, k, \epsilon_{\mathsf{AMD}})$*-AMD code. Formally, we define the circuit* $G_{\mathsf{mul}} : \mathbb{F}^k \times \mathbb{F}^k \to \mathbb{F}^k$ *as follows. G on the inputs* $(\mathbf{x}, \mathbf{y})$ *performs the following:*

1. *Compute* $(b, x') \leftarrow \mathsf{Dec}(\mathbf{x})$.
2. *Compute* $(b', y') \leftarrow \mathsf{Dec}(\mathbf{y})$.
3. *Compute* $f \leftarrow r_1 b + r_2 b'$ *where* r_1, r_2 *are random field elements.*
4. *Compute* $z \leftarrow x' \cdot y'$.
5. *Output* $(\mathsf{Enc}(z), f)$.

Similarly, we define the circuits $G_{\mathsf{add}} : \mathbb{F}^k \times \mathbb{F}^k \to \mathbb{F}^k$, $G_{\mathsf{sub}} : \mathbb{F}^k \times \mathbb{F}^k \to \mathbb{F}^k$ the same way as G_{mul} except that step 4 is replaced with $z \leftarrow x' + y'$ and $z \leftarrow x' - y'$ respectively.

Such components can be used in a straightforward way to obtain AMD circuits: first, replace every addition, subtraction and multiplication gate in C with G_{add}, G_{sub} and G_{mul} respectively. Next, append to each output gate an AMD decoder circuit Dec to perform the output decoding. Finally, combine the error flags of all the Dec circuits such that in case one of the decodings fails, the output of the entire circuit will be random. Since $G_{\mathsf{add}}, G_{\mathsf{sub}}, G_{\mathsf{mul}}$ verify their inputs and encode the outputs, the security of the AMD code guarantees that any additive attack on the internal wires of C will be caught with relatively high probability. Formally,

Construction 3.10. *Let $C : \mathbb{F}^n \to \mathbb{F}^k$ be a circuit containing only addition and multiplication gates and let $(\mathsf{Enc}, \mathsf{Dec})$ be the private $(1, k, \epsilon_{\mathsf{AMD}})$-AMD code used in Construction 3.9. We build $\widehat{C} : \mathbb{F}^n \to \mathbb{F}^k$ over the gate set $\mathcal{G} = \{G_{\mathsf{add}}, G_{\mathsf{sub}}, G_{\mathsf{mul}}, \mathsf{Dec}, \mathsf{Enc}\}$ in the following way:*

- *For every input gate g^c of C, $\widehat{C}$ will compute $x'_c \leftarrow \mathsf{Enc}(x_c)$ where x_c is the input corresponding to g^c.*
- *Replace the gates of C in topological order. For each gate g^c with inputs g^a and g^b, replace g^c with G_{add}, G_{sub} and G_{mul} based on the gate type of g^c. Let G^c be the resulting gate, connect the inputs of G^c to the first outputs of G^a and G^b.*
- *For each output gate g^c connect a circuit Dec to the first output of G^c.*
- *Let m be the number of gates from $\mathcal{G}$ used so far and let f_i be the second output of the i-th gate from $\mathcal{G}$ used so far. Generate $\mathbf{r} \in \mathbb{F}^m$ uniformly at random and compute $f \leftarrow r_i f_i$.*
- *Generate a vector $\mathbf{r}' \in \mathbb{F}^k$ uniformly at random.*
- *Let z_i be the first output of the i-th Dec gate. Output $(z_1, \cdots, z_k) + f\mathbf{r}'$.*

Theorem 3.5. *For any circuit $C : \mathbb{F}^n \to \mathbb{F}^k$ the randomized circuit $\widehat{C} : \mathbb{F}^n \to \mathbb{F}^k$ constructed in Construction 3.10 over the gate set $\mathcal{G}$ is an $1 - \frac{(1-\epsilon_{\mathsf{AMD}})(|\mathbb{F}|-1)}{|\mathbb{F}|}$-secure implementation of C. Moreover, $\widehat{C} = O(|C|)$.*

Proof. The completeness and complexity properties easily follow from the construction of $\widehat{C}$. To show the additive-attack security property, consider any additive attack A and let $\widetilde{C} \leftarrow A(\widehat{C})$. We split the proof into two cases.

- A attacks one of the inputs of some gate G of type G_{mul}, G_{add}, G_{sub} and Dec. Let G_i be the maximal attacked by A (in some topological ordering of the gates of $\widehat{C}$). By the additive robustness of $(\mathsf{Enc}, \mathsf{Dec})$ we have that the second output f_i of G_i is not zero with probability $(1 - \epsilon_{\mathsf{AMD}})$. This causes f to be non-zero with probability of $\frac{(1-\epsilon_{\mathsf{AMD}})(|\mathbb{F}|-1)}{|\mathbb{F}|}$. Thus we have that for any $\mathbf{x} \in \mathbb{F}^n$ it holds that
$$SD\left(\widetilde{C}(\mathbf{x}), U_k\right) \leq 1 - \frac{(1 - \epsilon_{\mathsf{AMD}})(|\mathbb{F}| - 1)}{|\mathbb{F}|}.$$
- A does not attack any of the inputs of gates of type G_{mul}, G_{add}, G_{sub} and Dec in $\widehat{C}$. Let $\mathbf{a}^{\mathsf{in}}$ be A restricted to the wires carrying $\mathbf{x}$ to the Enc circuits. In addition, for any i let $a_i^{\mathsf{out}} \in \mathbb{F}$ be A restricted to the second output the gate Dec $\widetilde{C}$ that is outputting the i-th output. Let $\mathcal{A}^{\mathsf{out}}$ be the distribution over $\mathbb{F}^k$ defined as follows: for all $1 \leq i \leq k$ if the wire carrying f into the circuit outputting z_i is attacked then $\mathcal{A}_i^{\mathsf{out}} = U$. Otherwise, $\mathcal{A}_i^{\mathsf{out}} = a_i^{\mathsf{out}}$. Thus we have that for any $\mathbf{x} \in \mathbb{F}^n$ it holds that
$$SD\left(\widetilde{C}(\mathbf{x}), C\left(\mathbf{x} + \mathbf{a}^{\mathsf{in}}\right) + \mathcal{A}^{\mathsf{out}}\right) = 0.$$

■

Corollary 3.1. *There exists a finite gate set $\mathcal{G}$ such that for any arithmetic circuit C over some finite field $\mathbb{F}$, there exists an arithmetic circuit $\widehat{C}$ of size $O(|C|)$ over $\mathcal{G}$ such that $\widehat{C}$ is an ϵ-secure implementation of C, where $\epsilon = 1 - \frac{(|\mathbb{F}|-1)^2}{|\mathbb{F}|^2}$.*

Proof. We use the Construction 3.10 and a private $(1, k, \frac{1}{|\mathbb{F}|})$-AMD code from Corollary 2.1. ■

3.3.2 Additive-attack security for arbitrary circuits

The tamper-proof components of the previous construction can be eliminated in a natural way by first implementing each component using a (constant-size) arithmetic circuit and then applying the construction for low-degree circuits to protect this circuit against additive attacks. Security is implied by the following composition theorem.

Construction 3.11. *Let $\mathbb{F}$ be a field, $C : \mathbb{F}^n \to \mathbb{F}^k$ be an arithmetic circuit, $C' : \mathbb{F}^n \to \mathbb{F}^k$ be circuit over some gate set $\mathcal{G}$ containing m gates $G_1, \ldots, G_m$ from $\mathcal{G}$ such that C' is an ϵ-secure implementation of C. In addition, for all $1 \leq i \leq m$ let $\widehat{G}_i$ be an ϵ_i-secure implementation of G_i. Consider the circuit $\widehat{C} : \mathbb{F}^n \to \mathbb{F}^k$ over the gate set $\{+, -, \times\}$ where the gate G_i is replaced with $\widehat{G}_i$ for every i.*

Theorem 3.6. *For any circuit $C : \mathbb{F}^n \to \mathbb{F}^k$ the circuit $\widehat{C} : \mathbb{F}^n \to \mathbb{F}^k$ constructed in Construction 3.11 is an $(\epsilon + \sum_{i=1}^{m} \epsilon_i)$-secure implementation of C.*

Proof. The completeness property easily follows from the construction of $\widehat{C}$. To show the additive-attack security property, consider any additive attack A. Let $\widetilde{C} \leftarrow A(C)$, and let A_i be A restricted to $\widehat{G}_i$. By the security property of $\widehat{G}_i$ it holds that there exist $\mathbf{a}^{\mathsf{in}}_{\widehat{G}_i}$ and a distribution $\mathcal{A}^{\mathsf{out}}_{\widehat{G}_i}$ such that for all $\mathbf{x}$ it holds that

$$SD\left(\widetilde{G_i}(\mathbf{x}), G_i(\mathbf{x} + \mathbf{a}^{\mathsf{in}}_{\widehat{G}_i}) + \mathcal{A}^{\mathsf{out}}_{\widehat{G}_i}\right) \leq \epsilon_i,$$

where $\widetilde{G_i} \leftarrow A_i(\widehat{G_i})$. Using Claim 2.1 we have that all the $\mathbf{a}^{\mathsf{in}}_{\widehat{G}_i}$'s and $\mathcal{A}^{\mathsf{out}}_{\widehat{G}_i}$'s induce an additive attack A' on C' such that for all $\mathbf{x} \in \mathbb{F}^n$

$$SD(\widetilde{C}(\mathbf{x}), \widetilde{C}'(\mathbf{x})) \leq \sum_{i=1}^{m} \epsilon_i$$

where $\widetilde{C}' \leftarrow A'(C')$.

By the ϵ-security of C' there exist $\mathbf{a}^{\mathsf{in}} \in \mathbb{F}^n$ and a distribution $\mathcal{A}^{\mathsf{out}}$ over $\mathbb{F}^k$ such that for all $\mathbf{x} \in \mathbb{F}^n$

$$SD(\widetilde{C}'(\mathbf{x}), C(\mathbf{x} + \mathbf{a}^{\mathsf{in}}) + \mathcal{A}^{\mathsf{out}}) \leq \epsilon$$

where $\widetilde{C}' \leftarrow A'(C')$. Thus, using Claim 2.2 we have that

$$SD(\widetilde{C}(\mathbf{x}), C(\mathbf{x} + \mathbf{a}^{\mathsf{in}}) + \mathcal{A}^{\mathsf{out}}) \leq \epsilon + \sum_{i=1}^{m} \epsilon_i.$$

■

Theorem 3.7 below states that any circuit C can be compiled into a circuit $\widehat{C}$ that is secure against additive attacks. Moreover, the size of $\widehat{C}$ is $O(|C|)$.

Theorem 3.7. *For any field $\mathbb{F}$ and an arithmetic circuit $C : \mathbb{F}^n \to \mathbb{F}^k$ there exists a circuit $\widehat{C} : \mathbb{F}^n \to \mathbb{F}^k$ of size $O(|\widehat{C}|)$ such that $\widehat{C}$-is an ϵ-secure implementation of C where $\epsilon = O\left(|C|/|\mathbb{F}|\right)$.*

Proof. Instantiate Construction 3.11 with the transformations from Corollary 3.1 and Construction 3.8. ■

We now state the following theorem about security against additive attack with a tamper-proof encoder and decoder.

Theorem 3.8. *Let n, k be positive integers and $\mathbb{F}$ be a finite field. Then, there exist a randomized input encoder circuit $\mathsf{Enc} : \mathbb{F}^n \to \mathbb{F}^{n'}$ of size $O(n)$ and an output decoder circuit $\mathsf{Dec} : \mathbb{F}^{k'} \to \mathbb{F} \times \mathbb{F}^k$ of size $O(k)$ such that the following holds. For any arithmetic circuit $C : \mathbb{F}^n \to \mathbb{F}^k$ there exists a randomized circuit $\widehat{C} : \mathbb{F}^{n'} \to \mathbb{F}^{k'}$ of size $O(|C|)$, such that for any additive attack A it holds that*

- *Perfect completeness. For any input $\mathbf{x} \in \mathbb{F}^n$, we have that*

$$\Pr[\mathsf{Dec}(\widehat{C}(\mathsf{Enc}(\mathbf{x}))) = (0, C(\mathbf{x}))] = 1.$$

- *Additive-attack correctness. For any input $\mathbf{x} \in \mathbb{F}^n$, we have that*

$$\Pr\left[\widetilde{C} \leftarrow A(C) : \mathsf{Dec}(\widetilde{C}(\mathsf{Enc}(\mathbf{x}))) \notin \mathsf{ERR} \cup \{(0, C(\mathbf{x}))\}\right] = O(|C|/|\mathbb{F}|).$$

Proof. Assume without loss of generality that $n = k$ and let $(\mathsf{Enc}, \mathsf{Dec})$ be an $(n, n', |C|/|\mathbb{F}|)$-AMD code. Define the circuit $C' : \mathbb{F}^{n'} \to \mathbb{F}^{n'}$ that on input $\mathbf{x} \in \mathbb{F}^n$ performs the following:

1. Compute $(b, \mathbf{x}') \leftarrow \mathsf{Dec}(\mathbf{x})$.
2. Compute $\mathbf{z} \leftarrow C(\mathbf{x}')$.
3. Output $\mathsf{Enc}(\mathbf{z}) + b\mathbf{r}$ where $\mathbf{r} \in \mathbb{F}^{n'}$ is a random vector.

The circuit $\widehat{C}$ is obtained by taking an $O(|C|/|\mathbb{F}|)$ secure implementation of C'. ■

4 AMD circuits over small finite fields

4.1 Correctness with constant error probability without a decoder

The construction of Section 3.3 is secure only for large finite fields, as discussed in Section 3.2.4. Here, we present a construction that has achieves the weaker notion of *additive-attack correctness with a decoder* (see Definition 1.2), with constant error probability that does not depend on the underlying field size. In Section 4.2.1 we will show how to amplify its correctness.

We sometimes would like to consider a notion of *additive-attack correctness* that is stronger than *additive-attack correctness with a decoder* as defined in Definition 1.2, yet weaker than full additive-attack security as defined in Definition 1.1. In this intermediate notion, $\widehat{C}$ should detect, with high probability, any additive attack made on it that does not correspond to an additive attack on the inputs and outputs of C. When such an attack is detected, we require that $\widehat{C}$ will set a special error flag to be non-zero. Formally,

Definition 4.1 (Additive-attack correctness without decoder)**.** *Let $\mathbb{F}$ be a finite field and let $f : \mathbb{F}^n \to \mathbb{F}^k$. We say that a randomized circuit $\widehat{C} : \mathbb{F}^n \to \mathbb{F}^{k+1}$ is an ϵ-correct implementation of f without a decoder if the following holds:*

- Perfect completeness. *For all $\mathbf{x} \in \mathbb{F}^n$, it holds that*

$$\Pr[\widehat{C}(\mathbf{x}) = (0, f(\mathbf{x}))] = 1$$

where the probability is taken over the randomness of $\widehat{C}$.

- Additive-attack correctness. *For any additive attack A, there exists $\mathbf{a}^{\mathsf{in}} \in \mathbb{F}^n$ and $\mathbf{a}^{\mathsf{out}} \in \mathbb{F}^k$ such that for all $\mathbf{x} \in \mathbb{F}^n$ it holds that*

$$\Pr\left[\widetilde{C} \leftarrow A(\widehat{C}) \ : \widetilde{C}(\mathbf{x}) \notin \mathsf{ERR} \cup \{(0, f(\mathbf{x} + \mathbf{a}^{\mathsf{in}}) + \mathbf{a}^{\mathsf{out}})\} \right] \leq \epsilon.$$

where the probability is taken over the randomness of $\widehat{C}$. We say that $\widehat{C}$ is an ϵ-correct implementation of a (deterministic or randomized) circuit C without a decoder if $\widehat{C}$ is an ϵ-correct implementation of the function f computed by C without a decoder.

4.1.1 Additively correct scalar-by-vector multiplier

In this section we consider the task of multiplying a vector $\mathbf{v} \in \mathbb{F}^n$ for some positive integer n and a finite field $\mathbb{F}$ by a field element $c \in \mathbb{F}$. That is, for every positive integer n we would like to design n-scalar-by-vector multiplier circuit C_n that gets as input a vector $\mathbf{v} \in \mathbb{F}^n$ and a scalar $x \in \mathbb{F}$ and outputs $z = \mathbf{v}x$. In this section we present an $\frac{1}{|\mathbb{F}|}$-correct implementation of C_n without a decoder for any positive integer n.

Notice that for the case where $n = 1$ this task can be easily achieved using a single multiplication gate. This does not holds for larger values of n. For example, for $n = 2$ we have that $z_1 = v_1 x$ and $z_2 = v_2 x$. However, consider the attack where $\widetilde{z}_1 = (x + a)v_1$ and $\widetilde{z}_2 = xv_2$ for some $a \neq 0$. Since each coordinate of $\mathbf{v}$ got multiplied by a different value this attack does not correspond to any attack on the inputs and outputs of the circuit. We thus show the following construction of a correct implementation of C_n without a decoder for any positive integer n. The main idea of the construction is similar to the randomized algorithm of Freivalds [Fre77] for matrix multiplication verification.

Construction 4.1. *Let $\mathbb{F}$ be a finite field and let n be positive integer. Consider the circuit $\widehat{C}$ defined as follows: On input $(\mathbf{v}, x)$, where $\mathbf{v} \in \mathbb{F}^n$, $\mathbf{v} = (v_1, \cdots, v_n)$, and $x \in \mathbb{F}$, the circuit $\widehat{C}$ does the following:*

1. *Generate a vector $\mathbf{r} \in \mathbb{F}^m$ uniformly at random.*
2. *For all $1 \leq i \leq n$, compute $z_i = x \cdot v_i$ and define $\mathbf{z} = z_1, \cdots, z_n$.*
3. *Compute $q = \mathbf{r} \cdot \mathbf{z}$.*
4. *Compute $y = \mathbf{r} \cdot \mathbf{v}$.*
5. *Compute $q' = x \cdot y$.*
6. *Compute $f = q - q'$.*
7. *Output $(f, \mathbf{z})$.*

Lemma 4.1. *The circuit $\widehat{C}$ constructed in Construction 4.2 is an ϵ-correct implementation of an n-scalar-by-vector multiplier without a decoder for $\epsilon = \frac{1}{|\mathbb{F}|}$.*

Proof. The completeness property easily follows from the construction of $\widehat{C}$. To show the additive-attack correctness property, consider any additive attack A on $\widehat{C}$. Let $x \in \mathbb{F}$ and $\mathbf{v} \in \mathbb{F}^n$. By the definition of $\widehat{C}$ it holds that the subcircuits computing q' and the coordinates $z_1, \ldots, z_n$ of z in steps 2 and 5 can be expressed in the following equations:

$$z_i = x \cdot v_i,$$

$$q' = x \cdot \left(\sum_{i=1}^{n} r_i \cdot v_i \right).$$

By Lemma 2.1 we can assume without loss of generality that A only attacks the inputs of multiplication and output gates inside $\widetilde{C}$. That is, A causes the subcircuits computing $z_1, \ldots, z_n$ and q' inside $\widetilde{C}$ to compute the following

$$z_i = (x + A_{x,z_i})(v_i + A_{v_i,z_i}) = xv_i + xA_{v_i,z_i} + A_{x,z_i}v_i + A_{x,z_i}A_{v_i,z_i}.$$

$$\begin{aligned}
q' &= (x + A_{x,q'}) \cdot \left(A_{y,q'} + \sum_{i=1}^{n} (r_i + A_{r_i,y}) \cdot (v_i + A_{v_i,y}) \right) \\
&= (x + A_{x,q'}) \cdot \left(\sum_{i=1}^{n} (r_i v_i + r_i A_{v_i,y} + A_{r_i,y} v_i + A_{r_i,y} A_{v_i,y}) \right) + A_{y,q'}(x + A_{x,q'}) \\
&= \sum_{i=1}^{n} r_i (x v_i + A_{x,q'} v_i + x A_{v_i,y} + A_{x,q'} A_{v_i,y}) \\
&+ (x + A_{x,q'}) \cdot \sum_{i=1}^{n} (A_{r_i,y} v_i + A_{r_i,y} A_{v_i,y}) + A_{y,q'}(x + A_{x,q'}).
\end{aligned}$$

Let a be A restricted to output of the gate computing f. By the definition of $\widetilde{C}$ it holds that

$$\Pr\left[\widetilde{C} \leftarrow A(\widehat{C}) \ : \widetilde{C}(\mathbf{v}, x) \notin \mathsf{ERR} \right] \leq \Pr_{\mathbf{r}} \left[f - a = 0 \right].$$

We split the proof of the theorem into two cases:

- There exists i such that

  $$x A_{v_i,z_i} + A_{x,z_i} v_i + A_{x,z_i} A_{v_i,z_i} + A_{z_i,q} - A_{x,q'} v_i - x A_{v_i,y} - A_{x,q'} A_{v_i,y} \neq 0.$$

  Recall that $\mathbf{A}_{\mathbf{r},q} \in \mathbb{F}^m$ and $\mathbf{A}_{\mathbf{z},q} \in \mathbb{F}^n$ are the restriction of A to the wires carrying $\mathbf{r}$ to q and $\mathbf{z}$ to q respectively. Notice that A causes the subcircuit computing f to compute the following:

  $$\begin{aligned}
  f &= q - q' \\
  &= (\mathbf{r} + \mathbf{A}_{\mathbf{r},q})(\mathbf{z} + \mathbf{A}_{\mathbf{z},q}) - q' \\
  &= \sum_{i=1}^{n} (r_i + A_{r_i,q})(z_i + A_{z_i,q}) - q' \\
  &= \sum_{i=1}^{n} (r_i + A_{r_i,q})(x v_i + x A_{v_i,z_i} + A_{x,z_i} v_i + A_{x,z_i} A_{v_i,z_i} + A_{z_i,q}) \\
  &\quad - \sum_{i=1}^{n} r_i (x v_i + A_{x,q'} v_i + x A_{v_i,y} + A_{x,q'} A_{v_i,y}) \\
  &\quad - (x + A_{x,q'}) \cdot \sum_{i=1}^{n} (A_{r_i,y} v_i + A_{r_i,y} A_{v_i,y}) + A_{y,q'}(x + A_{x,q'}) \\
  &= \sum_{i=1}^{n} r_i (x A_{v_i,z_i} + A_{x,z_i} v_i + A_{x,z_i} A_{v_i,z_i} + A_{z_i,q} - A_{x,q'} v_i - x A_{v_i,y} - A_{x,q'} A_{v_i,y}) \\
  &\quad + \sum_{i=1}^{n} A_{r_i,q} (x v_i + x A_{v_i,z_i} + A_{x,z_i} v_i + A_{x,z_i} A_{v_i,z_i} + A_{z_i,q}) \\
  &\quad - (x + A_{x,q'}) \cdot \sum_{i=1}^{n} (A_{r_i,y} v_i + A_{r_i,y} A_{v_i,y}) + A_{y,q'}(x + A_{x,q'})
  \end{aligned}$$

Fix a selection for all the coordinates of r but the ith coordinate. By assumption it holds that

$$xA_{v_i,z_i} + A_{x,z_i}v_i + A_{x,z_i}A_{v_i,z_i} + A_{z_i,q} - A_{x,q'}v_i - xA_{v_i,y} - A_{x,q'}A_{v_i,y} \neq 0.$$

Thus, f is of the form $f = c_1 r_i + c_2$ where $c_1, c_2 \in \mathbb{F}$ and $c_1 \neq 0$. Thus we have that

$$\Pr\left[\ \widetilde{C} \leftarrow A(\widehat{C})\ \ : \widetilde{C}(\mathbf{v}, x) \notin \mathsf{ERR}\right] \leq \Pr_{\mathbf{r}}\left[f - a = 0\right] \leq \Pr_{r_i}\left[c_1 r_i + c_2 - a = 0\right] \leq \frac{1}{|\mathbb{F}|}.$$

- Otherwise, for all i it holds that

$$xA_{v_i,z_i} + A_{x,z_i}v_i + A_{x,z_i}A_{v_i,z_i} + A_{z_i,q} - A_{x,q'}v_i - xA_{v_i,y} - A_{x,q'}A_{v_i,y} = 0.$$

Then, for all i it holds that

$$\begin{aligned} z_i &= xv_i + xA_{v_i,z_i} + A_{x,z_i}v_i + A_{x,z_i}A_{v_i,z_i} \\ &= xv_i + A_{x,q'}v_i + xA_{v_i,y} + A_{x,q'}A_{v_i,y} - A_{z_i,q} \\ &= (x + A_{x,q'})(v_i + A_{v_i,y}) - A_{z_i,q}. \end{aligned}$$

Let $\mathbf{a}^{\mathsf{in}} \in \mathbb{F}^n$ be a vector such that $a_i^{\mathsf{in}} = A_{v_i,y}$ and let $\mathbf{a}^{\mathbf{z}}$ be A restricted to the wires carrying z to the output. Define the vector $\mathbf{a}^{\mathsf{out}} \in \mathbb{F}^n$ be a vector such that $\mathbf{a}_i^{\mathsf{out}} = a_i^{\mathbf{z}} + A_{z_i,q}$. We have that

$$\Pr\left[\ \widetilde{C} \leftarrow A(\widehat{C})\ \ : \widetilde{C}(v, x) \notin \mathsf{ERR} \cup \{(0, (\mathbf{v} + \mathbf{a}^{\mathsf{in}})(x + A_{x,q'}) + \mathbf{a}^{\mathsf{out}})\}\right] = 0.$$

■

4.1.2 Additively correct matrix-by-vector multiplier

Let $\mathbb{F}$ be a finite field and n, m be positive integers. An $(m \times n)$-multiplier over $\mathbb{F}$ is a circuit $C_{m\times n}$ that gets as input a vector $\mathbf{w} \in \mathbb{F}^n$ and a matrix $M \in \mathbb{F}^{m\times n}$. The circuit then outputs $z = M\mathbf{w}$. In this section we present a $(\frac{1-\epsilon}{|\mathbb{F}|} + \epsilon)$-correct implementation of $C_{m\times n}$ without a decoder as defined in Definition 4.1 above using an ϵ-correct vector-by-scalar multiplier. By combining this construction with Construction 4.1 we will obtain a $(\frac{2|\mathbb{F}|-1}{|\mathbb{F}|^2})$-correct implementation of $C_{m\times n}$

Notice that in the case where $m = 1$ the circuit $C_{1\times n}$ computes the inner product of $\mathbf{w}$ and M over $\mathbb{F}$. Since in this case the circuit $C_{1\times 1}$ has multiplicative depth 1 and no internal fan-out, the naive implementation of $\widehat{C}$ (coordinate-wise multiplication followed by summing up the result) has the desired additive-attack correctness property. This does not hold for larger values of m.[8] We thus show the following construction of a correct implementation of $C_{m\times n}$ without a decoder for any $n, m \in \mathbb{N}$.

Construction 4.2 (Matrix multiplier)**.** *Let $\mathbb{F}$ be a finite field and let n, m be positive integers. In addition, let C' be an ϵ'-correct implementation of an m-scalar-by-vector without a decoder. Consider the circuit $\widehat{C}$ defined as follows. On input $(M, \mathbf{w})$, where $\mathbf{w} \in \mathbb{F}^n$ and $(M_1, \ldots, M_n)$ are the columns of an $m \times n$ matrix M over $\mathbb{F}$, the circuit $\widehat{C}$ does the following:*

1. *For every $1 \leq i \leq n$ compute $(f_i, \mathbf{y}^i) = C'(M_i, w_i)$.*[9]

[8] For example, for $m = 2$ we have that $z_1 = M_1\mathbf{w}$ and $z_2 = M_2\mathbf{w}$ where M_1 and M_2 are the rows of the matrix M. However, consider the attack where $\widetilde{z}_1 = M_1\mathbf{w}$ and $\widetilde{z}_2 = M_2(\mathbf{w} + \mathbf{a})$ for some $\mathbf{a} \neq 0$. Since each row of the matrix got multiplied by a different value this attack does not correspond to any attack on the inputs and outputs of $C_{2\times n}$.

[9] We observe that for the construction to hold it enough to run all the different copies of C' with the same randomness.

2. *For every* $1 \leq j \leq m$ *compute* $z_j = \sum_{i=1}^{m} y_j^i$

3. *For every* $1 \leq i \leq m$ *compute* $f = \sum_{i=1}^{m} r_i f_i$ *where* $r_i \in \mathbb{F}$ *is a random field element.*

4. *Define* $\mathbf{z} = z_1, \cdots, z_m$.

5. *Output* $(f, \mathbf{z})$.

Lemma 4.2. *The circuit* $\widehat{C}$ *constructed in Construction 4.2 is an* ϵ*-correct implementation of an* (n, m)*-multiplier without a decoder for* $\epsilon = 1 - \frac{(1-\epsilon')(|\mathbb{F}|-1)}{|\mathbb{F}|} = \frac{1-\epsilon'}{|\mathbb{F}|} + \epsilon'$.

Proof. The completeness property easily follows from the construction of $\widehat{C}$. As for the additive-attack correctness property, since C' is a correct implementation of an n-scalar-by-vector multiplier we have that for all i any attack on C'_i is equivalent to an attack on the inputs and outputs of C'_i or will cause f_i to become non-zero with probability at least $1 - \epsilon'$. Let A be an attack on $\widehat{C}$.

If there exists i such that f_i is non-zero with probability of at least $1 - \epsilon'$ we have that f is non-zero with probability of at least $\frac{(1-\epsilon')(|\mathbb{F}|-1)}{|\mathbb{F}|}$ and the theorem follows.

Otherwise, we have that for any i the attack on C'_i is equivalent to an attack on the inputs and outputs of C'_i. By the design of $\widehat{C}$ we have that for all i the attack on the inputs of C'_i is equivalent to an attack on the inputs of $\widehat{C}$. Similarly, since the circuits computing $\mathbf{z}$ only contain addition gates, by Lemma 2.1 we have that for all i the attack on the outputs of C'_i is equivalent to an attack on the outputs of $\widehat{C}$. ■

Remark 4.1. Construction 4.2 above uses $n + m$ random field elements in order to correctly implement an $(m \times n)$-multiplier. A more efficient (but harder to analyze) construction can be obtained by combining Constructions 4.1 and 4.2 above. That it, in order to multiply an $m \times n$ matrix M by a vector $\mathbf{x} \in \mathbb{F}^n$ one can compute $\mathbf{z} = M\mathbf{x}$; $q = (\mathbf{r}^T M)\mathbf{x}$; $f = \mathbf{r}^T\mathbf{z} - q$ and output $(f, \mathbf{z} + f\mathbf{r}')$ where $\mathbf{r}$ and $\mathbf{r}'$ are generated uniformly from $\mathbb{F}^n$ and $\mathbb{F}^m$ respectively.

4.1.3 Additively correct circuits with quadratic overhead

In this section we construct an $|\mathbb{F}|^{-\Omega(1)}$-correct implementation with quadratic overhead for arbitrary circuits containing only bilinear gates over any field $\mathbb{F}$. The construction will rely on a simplified version of the Hadamard PCP of [ALM+92]. In the standard PCP setting the prover provides a proof string to a verifier. The verifier, in turn, verifies that the provided proof string has some desired additive properties. It is assumed that the prover can be dishonest and that the verifier is always honest.

We prove that when we limit the adversary to additive attacks, we in fact can allow attacks on the *verifier* as well as on the prover without losing soundness. Thus, we obtain a way of verifying that the computation of the circuit was done correctly.

We will use special gates called *bilinear* gates. Each such gate gets as input an unbounded number of wires from the left and from the right and has one output wire. For a bilinear gate v denote by $\mathsf{left}(v)$ and $\mathsf{right}(v)$ the numbers of the left and right input wires respectively of v. The output wire w_v of v is defined as $w_v = \left(\sum_{i \in \mathsf{left}(v)} w_i\right)\left(\sum_{j \in \mathsf{right}(v)} w_j\right)$. In the sequel we will assume that C only has gates of the form $w_v = \left(\sum_{i \in \mathsf{left}(v)} w_i\right)\left(\sum_{j \in \mathsf{right}(v)} w_j\right)$ for some gate v. We can make this assumption without loss of generality since by setting $|\mathsf{left}(v)| = 2$, $|\mathsf{right}(v)| = 1$ and $y_j = 1$ where $j \in \mathsf{right}(v)$ we obtain an addition gate and by setting $|\mathsf{left}(v)| = |\mathsf{right}(v)| = 1$ we obtain a multiplication gate. Thus, every arithmetic circuit can be implemented using such *bilinear* gates (see Remark 3.1 regarding the support of constant gates).

Before presenting construction, we notice that for degree 2 circuits that does not have any fanout, we can assume without loss of generality that only the inputs and outputs of C are attacked since the

attack on the internal wires of C can be pulled "upstream" through the addition gates until the inputs of C. Formally,

Lemma 4.3. *Let $C(x)$ be a randomized arithmetic circuit of degree 2 that does not have any internal fanout and* one *gates. In addition, let A be an additive attack. We claim that there exists an additive attack A' that only attacks the inputs and outputs of C such that for all x it holds that*

$$\widetilde{C}(x) \equiv \widetilde{C}'(x)$$

where $\widetilde{C} \leftarrow A(C)$ and $\widetilde{C}' \leftarrow A'(C)$.

We now present our construction for additively secure circuits with quadratic overhead.

Construction 4.3. *Let $C : \mathbb{F}^n \to \mathbb{F}^k$ be a circuit containing t bilinear gates, $C_{3\times t}$ be an ϵ'-correct implementation of a $(3\times t)$-multiplier and $C_{2\times t^2}$ be an ϵ'-correct implementation of a $(2\times t^2)$-multiplier. Consider the circuit $\widehat{C} : \mathbb{F}^n \to \mathbb{F}^{k+1}$ defined as follows on input x:*

1. *Compute the circuit C on input $\mathbf{x}$ and obtain an output $\mathbf{z}$ together with a vector $\mathbf{w} \in \mathbb{F}^t$ representing the intermediate values of all the gate outputs during the computation of C on the input $\mathbf{x}$. We require that the last k entries of $\mathbf{w}$ will contain the outputs of the gates connected to the outputs of C.*
2. *Compute the vector $\mathbf{w}' \in \mathbb{F}^{t\times t}$ as follows: for any $1 \le i, j \le t$ set $w'_{i,j} = w_i \cdot w_j$.*
3. *Generate two random vectors $\mathbf{r}, \mathbf{s}$ from $\mathbb{F}^t$.*
4. *Compute the vector $\mathbf{q} \in \mathbb{F}^{t\times t}$ as follows: for any $1 \le i, j \le t$ set $q_{i,j} = r_i \cdot s_j$.*
5. *Generate a random vector $\mathbf{p} \in \mathbb{F}^t$.*
6. *Let $\mathbf{u} \in \mathbb{F}^k$ be the last k entries of $\mathbf{p}$. That is, for all $1 \le i \le k$ we define $u_i = p_{t-1+i}$.*
7. *For any $1 \le j, j' \le t$ define the set $I_{j,j'} = \{i : j \in \mathsf{left}(i) \text{ and } j' \in \mathsf{right}(i)\}$.*
8. *Generate the vector $\mathbf{q}' \in \mathbb{F}^{t\times t}$ as follows: for all $1 \le j, j' \le t$ set $q'_{j,j} = \sum_{i\in I_{j,j'}} p_i$.*
9. *Compute $(f_1, (v^{\mathbf{rw}}, v^{\mathbf{sw}}, v^{\mathbf{pw}})) \leftarrow C_{3\times t}(M, \mathbf{w})$ where M is a matrix whose rows are $\mathbf{r}, \mathbf{s}, \mathbf{p}$.*
10. *Compute $(f_2, (v^{\mathbf{qw}'}, v^{\mathbf{q}'\mathbf{w}'})) \leftarrow C_{2\times t^2}(M', \mathbf{w}')$ where M' is a matrix whose rows are $\mathbf{q}, \mathbf{q}'$.*
11. *Generate $r_1, r_2, r_3, r_4 \in \mathbb{F}$ uniformly at random.*
12. *Compute $f = r_1(v^{\mathbf{rw}} \cdot v^{\mathbf{sw}} - v^{\mathbf{qw}'}) + r_2(v^{\mathbf{pw}} - v^{\mathbf{q}'\mathbf{w}'} - \mathbf{z}^T\mathbf{u}) + r_3 f_1 + r_4 f_2$.*
13. *Output $(f, \mathbf{z})$.*

Theorem 4.1. *For any circuit C, the circuit $\widehat{C}$ as obtained via Construction 4.3 with respect to C above is a $1 - \frac{(1-\epsilon')^2(|\mathbb{F}|-1)^3}{|\mathbb{F}|^3}$-correct implementation of C without a decoder. Moreover, when using Construction 4.2 to construct $C_{3\times t}$ and $C_{2\times t^2}$ we obtain that $|\widehat{C}| = O(|C|^2)$.*

Proof. The completeness and complexity properties easily follow from the construction of $\widehat{C}$. To show the additive-attack correctness property, consider any additive attack A on $\widehat{C}$. Let $\mathbf{a}^{\mathbf{x}}$ be A restricted to the wires carrying the input $\mathbf{x}$ inside the vector $\mathbf{w}$ in C. In addition, let $\mathbf{a}^{\mathsf{out}}$ be A restricted to the output wires of C.

We start by claiming that by the design of $\widehat{C}$ we can assume without loss of generality that A does not attack the entire sub-circuit inside $\widehat{C}$ whose inputs are $v^{\mathbf{rw}}, v^{\mathbf{sw}}, v^{\mathbf{qw'}}, v^{\mathbf{pw}}, v^{\mathbf{q'w'}}, \mathbf{z}, \mathbf{u}, f_1, f_2, r_1, r_2, r_3, r_4$ and whose output is f in Step 12. Indeed, first notice that by Lemma 4.3 we can assume that only the inputs to this circuit are attacked. Next, observe that any attacks on the output of the randomness gates r_1, r_2, r_3 and r_4 does not change their output distribution. Finally, any attacks on the other input wires $(v^{\mathbf{rw}}, v^{\mathbf{sw}}, v^{\mathbf{qw'}}, v^{\mathbf{pw}}, v^{\mathbf{q'w'}}, \mathbf{z}, \mathbf{u})$ could be considered as part of $A^{C_{3\times t}}$ and $A^{C_{2\times t^2}}$.

Notice that by the additive-attack correctness of $C_{3\times t}$ and $C_{2\times t^2}$ we obtain that there exist attacks on the inputs of the multipliers $\mathbf{a^w}, \mathbf{a^r}, \mathbf{a^s}, \mathbf{a^p} \in \mathbb{F}^t$, $\mathbf{a}^{\mathbf{w'}}, \mathbf{a^q}, \mathbf{a}^{\mathbf{q'}} \in \mathbb{F}^{t\times t}$ and attacks on the output of the multipliers $b^{\mathbf{rw}}, b^{\mathbf{sw}}, b^{\mathbf{pw}}, b^{\mathbf{qw'}}, b^{\mathbf{q'w'}} \in \mathbb{F}$ such that the following holds. For any $\mathbf{w}, \mathbf{r}, \mathbf{s}, \mathbf{p} \in \mathbb{F}^t$ it holds that

$$\Pr\left[\widetilde{C}_{3\times t} \leftarrow A^{C_{3\times t}}(C_{3\times t}) : \widetilde{C}_{3\times t}(\mathbf{r}, \mathbf{s}, \mathbf{p}, \mathbf{w}) \in I_{3\times t}\right] \geq 1 - \epsilon' \tag{4}$$

where $A^{C_{3\times t}}$ is A restricted to $C_{3\times t}$ (including the input and output wires), $I_{3\times t} = \mathsf{ERR} \cup \{(0, \langle \mathbf{w} + \mathbf{a^w}, \mathbf{r} + \mathbf{a^r}\rangle + b^{\mathbf{rw}}, \langle \mathbf{w} + \mathbf{a^w}, \mathbf{s} + \mathbf{a^s}\rangle + b^{\mathbf{sw}}, \langle \mathbf{w} + \mathbf{a^w}, \mathbf{p} + \mathbf{a^p}\rangle + b^{\mathbf{pw}})\}$ and the probability is over the internal randomness of $\widetilde{C}_{3\times t}$.

Similarly, for any $\mathbf{w'}, \mathbf{q}, \mathbf{q'} \in \mathbb{F}^{t\times t}$ it holds that

$$\Pr\left[\widetilde{C}_{2\times t^2} \leftarrow A^{C_{2\times t^2}}(C_{2\times t^2}) : \widetilde{C}_{2\times t^2}(\mathbf{q}, \mathbf{q'}, \mathbf{w'}) \in I_{2\times t^2}\right] \geq 1 - \epsilon' \tag{5}$$

where $A^{C_{2\times t^2}}$ is A restricted to $C_{2\times t^2}$ (including the input and output wires), $I_{2\times t^2} = \mathsf{ERR} \cup \{(0, \langle \mathbf{w'} + \mathbf{a}^{\mathbf{w'}}, \mathbf{q} + \mathbf{a^q}\rangle + b^{\mathbf{qw'}}, \langle \mathbf{w'} + \mathbf{a}^{\mathbf{w'}}, \mathbf{q'} + \mathbf{a}^{\mathbf{q'}}\rangle + b^{\mathbf{q'w'}})\}$ and the probability is over the internal randomness of $\widetilde{C}_{2\times t^2}$.

Observe that the randomness used in Equations 4 and 5 is independent. Thus, combining Equations 4 and 5 we obtain that for any $\mathbf{w}, \mathbf{r}, \mathbf{s}, \mathbf{p} \in \mathbb{F}^t$ and for any $\mathbf{w'}, \mathbf{q}, \mathbf{q'} \in \mathbb{F}^{t\times t}$ it holds that

$$\Pr\left[\begin{array}{c} \widetilde{C}_{3\times t} \leftarrow A^{C_{3\times t}}(C_{3\times t}) \\ \widetilde{C}_{2\times t^2} \leftarrow A^{C_{2\times t^2}}(C_{2\times t^2}) \end{array} : \widetilde{C}_{3\times t}(\mathbf{r}, \mathbf{s}, \mathbf{p}, \mathbf{w}) \in I_{3\times t} \wedge \widetilde{C}_{2\times t^2}(\mathbf{q}, \mathbf{q'}, \mathbf{w'}) \in I_{2\times t^2}\right] \geq (1 - \epsilon')^2. \tag{6}$$

where the probability is over the internal randomness of $\widetilde{C}_{3\times t}$, $\widetilde{C}_{2\times t^2}$ and as defined above,

- $A^{C_{3\times t}}$ is A restricted to $C_{3\times t}$ and $A^{C_{2\times t^2}}$ is A restricted to $C_{2\times t^2}$,
- $I_{3\times t} = \mathsf{ERR} \cup \{(0, \langle \mathbf{w} + \mathbf{a^w}, \mathbf{r} + \mathbf{a^r}\rangle + b^{\mathbf{rw}}, \langle \mathbf{w} + \mathbf{a^w}, \mathbf{s} + \mathbf{a^s}\rangle + b^{\mathbf{sw}}, \langle \mathbf{w} + \mathbf{a^w}, \mathbf{p} + \mathbf{a^p}\rangle + b^{\mathbf{pw}})\}$,
- $I_{2\times t^2} = \mathsf{ERR} \cup \{(0, \langle \mathbf{w'} + \mathbf{a}^{\mathbf{w'}}, \mathbf{q} + \mathbf{a^q}\rangle + b^{\mathbf{qw'}}, \langle \mathbf{w'} + \mathbf{a}^{\mathbf{w'}}, \mathbf{q'} + \mathbf{a}^{\mathbf{q'}}\rangle + b^{\mathbf{q'w'}})\}$.

In order to prove the theorem we will show that for any $\mathbf{x} \in \mathbb{F}^n$, whenever A did not raise the error flag, it is equivalent to an ideal attack (that only depends on A) on the inputs and outputs of $\widehat{C}$. Moreover, we will show that this ideal attack consists simply of the *inevitable* attack (on the inputs and outputs of C) plus the entries of $\mathbf{a^w}$ corresponding to the inputs of C. Formally, we will show that for all $\mathbf{x} \in \mathbb{F}^n$ it holds that

$$\Pr\left[\widetilde{C} \leftarrow A(\widehat{C}) : \widetilde{C}(x) \in \mathsf{ERR} \cup \{(0, C(\mathbf{x} + \mathbf{a}^{\mathsf{in}}) + \mathbf{a}^{\mathsf{out}})\}\right] \geq \frac{(1 - \epsilon')^2(|\mathbb{F}| - 1)^3}{|\mathbb{F}|^3}$$

where $\mathbf{a}^{\mathsf{in}} = \mathbf{a^x} + \mathbf{a}^{\mathbf{w},\mathsf{inp}}$ and $\mathbf{a}^{\mathbf{w},\mathsf{inp}} \in \mathbb{F}^n$ is vector of entries in $\mathbf{a^w}$ corresponding to the input wires of C.

In the sequel we will treat $\mathbf{w'}, \mathbf{a}^{\mathbf{w'}} \in \mathbb{F}^{t^2}$ both as vectors from $\mathbb{F}^{t^2}$ and as a $t \times t$ matrix over $\mathbb{F}$. Thus, we denote the $(i-1)t + j$th coordinate of $\mathbf{w'}$ and $\mathbf{a}^{\mathbf{w'}}$ as $a^{\mathbf{w'}}_{i,j}$ and $w'_{i,j}$.

Let $\mathbf{x} \in \mathbb{F}^n$. Recall that $\mathbf{w} \in \mathbb{F}^t$ is a vector representing the intermediate values of all the gates outputs during the commutation of C on input $\mathbf{x}$. In addition, recall that in the uncorrupted case $\mathbf{w'} = \mathbf{w} \times \mathbf{w}$. Notice that both $\mathbf{w}$ and $\mathbf{w'}$ are completely determined by $\mathbf{x}$ and A. We split the proof into two cases based on $\mathbf{w} + \mathbf{a^w}$ and $\mathbf{w'} + \mathbf{a}^{\mathbf{w'}}$.

1. It holds that
$$\mathbf{w}' + \mathbf{a}^{\mathbf{w}'} \neq (\mathbf{w} + \mathbf{a}^{\mathbf{w}}) \cdot (\mathbf{w} + \mathbf{a}^{\mathbf{w}})^T.$$
This means that the computation of $\mathbf{w}'$ was corrupted, and we will show that this corruption will be flagged by the consistency checks of $\mathbf{r}, \mathbf{s}, \mathbf{q}$ (assuming $C_{3\times t}$ and $C_{2\times t^2}$ where not themselves corrupted; otherwise, flags will be raised by $C_{3\times t}$ and $C_{2\times t^2}$).
Since $\mathbf{w}' + \mathbf{a}^{\mathbf{w}'} \neq (\mathbf{w} + \mathbf{a}^{\mathbf{w}}) \cdot (\mathbf{w} + \mathbf{a}^{\mathbf{w}})^T$ this means that there exist $1 \leq i', j' \leq t$ such that
$$w'_{i',j'} + a^{\mathbf{w}'}_{i',j'} \neq (w_{i'} + a^{\mathbf{w}}_{i'}) \cdot (w_{j'} + a^{\mathbf{w}}_{j'}).$$
Notice that A causes the subcircuit computing $\mathbf{q}$ to actually compute the following:
$$q_{i,j} = (r_i + A_{r_i,q_{i,j}}) \cdot (s_j + A_{s_j,q_{i,j}}).$$
From Equation 6 above it holds that
$$\Pr\left[(f_1 \neq 0 \vee f_2 \neq 0) \vee \begin{pmatrix} v^{\mathbf{rw}} = \langle \mathbf{w} + \mathbf{a}^{\mathbf{w}}, \mathbf{r} + \mathbf{a}^{\mathbf{r}} \rangle + b^{\mathbf{rw}} \wedge \\ v^{\mathbf{sw}} = \langle \mathbf{w} + \mathbf{a}^{\mathbf{w}}, \mathbf{s} + \mathbf{a}^{\mathbf{s}} \rangle + b^{\mathbf{sw}} \wedge \\ v^{\mathbf{qw}'} = \langle \mathbf{w}' + \mathbf{a}^{\mathbf{w}'}, \mathbf{q} + \mathbf{a}^{\mathbf{q}} \rangle + b^{\mathbf{qw}'} \end{pmatrix}\right] \geq (1 - \epsilon')^2. \tag{7}$$
In the sequel we will prove that the check $v_{\mathbf{rw}} \cdot v_{\mathbf{sw}} - v_{\mathbf{qw}'}$ in step 12 that is suppose to verify that $\mathbf{w}' + \mathbf{a}^{\mathbf{w}'} = (\mathbf{w} + \mathbf{a}^{\mathbf{w}}) \cdot (\mathbf{w} + \mathbf{a}^{\mathbf{w}})^T$ will fail with a constant probability causing f to become random. We start by proving that the probability over the selection of $\mathbf{r}, \mathbf{s}$ that $(\langle \mathbf{w} + \mathbf{a}^{\mathbf{w}}, \mathbf{r} + \mathbf{a}^{\mathbf{r}} \rangle + b^{\mathbf{rw}}) \cdot (\langle \mathbf{w} + \mathbf{a}^{\mathbf{w}}, \mathbf{s} + \mathbf{a}^{\mathbf{s}} \rangle + b^{\mathbf{sw}}) = (\langle \mathbf{w}' + \mathbf{a}^{\mathbf{w}'}, \mathbf{q} + \mathbf{a}^{\mathbf{q}} \rangle + b^{\mathbf{qw}'})$ is bounded by a universal constant. Indeed, this probability is bounded by the probability (over the selection of $\mathbf{r}, \mathbf{s}$) that the following equation holds:
$$\left(b^{\mathbf{rw}} + \sum_{i=1}^{t} (w_i + a^{\mathbf{w}}_i) \cdot (r_i + a^{\mathbf{r}}_i)\right) \cdot \left(b^{\mathbf{sw}} + \sum_{j=1}^{t} (w_j + a^{\mathbf{w}}_j) \cdot (s_j + a^{\mathbf{s}}_j)\right) \tag{8}$$
$$= b^{\mathbf{qw}'} + \sum_{i=1}^{t^2} \left(w'_i + a^{\mathbf{w}'}_i\right) \cdot \left(q_i + a^{\mathbf{q}}_i\right).$$
Equation 8 can be rewritten as
$$\begin{array}{ll} \sum_{(i,j)\neq(i',j')} \left((w_i + a^{\mathbf{w}}_i) \cdot (r_i + a^{\mathbf{r}}_i)\right) \cdot \left((w_j + a^{\mathbf{w}}_j) \cdot (s_j + a^{\mathbf{s}}_j)\right) & + \\ b^{\mathbf{sw}} \sum_{i\in[t]\setminus\{i'\}} \left((w_i + a^{\mathbf{w}}_i) \cdot (r_i + a^{\mathbf{r}}_i)\right) + b^{\mathbf{rw}} \sum_{j\in[t]\setminus\{j'\}} \left((w_j + a^{\mathbf{w}}_j) \cdot (s_j + a^{\mathbf{s}}_j)\right) & + \\ -\sum_{(i,j)\neq(i',j')} \left(w'_{i,j} + a^{\mathbf{w}'}_{i,j}\right) \cdot \left(q_{i,j} + a^{\mathbf{q}}_{i,j}\right) - b^{\mathbf{qw}'} & = \\ -(w_{i'} + a^{\mathbf{w}}_{i'}) \cdot (r_{i'} + a^{\mathbf{r}}_{i'}) \cdot (w_{j'} + a^{\mathbf{w}}_{j'}) \cdot (s_{j'} + a^{\mathbf{s}}_{j'}) & + \\ -(w_{i'} + a^{\mathbf{w}}_{i'}) \cdot (r_{i'} + a^{\mathbf{r}}_{i'}) \cdot b^{\mathbf{sw}} & + \\ -(w_{j'} + a^{\mathbf{w}}_{j'}) \cdot (s_{j'} + a^{\mathbf{s}}_{j'}) \cdot b^{\mathbf{rw}} & + \\ \left(w'_{i',j'} + a^{\mathbf{w}'}_{i',j'}\right) \cdot \left(q_{i',j'} + a^{\mathbf{q}}_{i',j'}\right). & \end{array} \tag{9}$$
We will show that for any selection of values to all the coordinates of $\mathbf{r}, \mathbf{s}$ but $r_{i'}$ and $s_{j'}$ there exists a selection of $r_{i'}$ and $s_{j'}$ such that Equation 9 does not hold. Indeed, fix some values to all the coordinates of $\mathbf{r}, \mathbf{s}$ but $r_{i'}$ and $s_{j'}$. Notice that the left hand side of Equation 9 does not depend on the values of $r_{i'}$ and $s_{j'}$. Denote by c the value of the left hand side of Equation 9. Thus, Equation 9 can be rewritten as
$$\begin{array}{lll} -\left(w_{i'} + a^{\mathbf{w}}_{i'}\right) \cdot \left(w_{j'} + a^{\mathbf{w}}_{j'}\right) \cdot \left(r_{i'} \cdot s_{j'} + r_{i'} \cdot a^{\mathbf{s}}_{j'} + a^{\mathbf{r}}_{i'} \cdot s_{j'} + a^{\mathbf{r}}_{i'} \cdot a^{\mathbf{s}}_{j'}\right) & + & \\ \left(w'_{i',j'} + a^{\mathbf{w}'}_{i',j'}\right) \cdot \left((r_i + A_{r_i,q_{i,j}}) \cdot (s_j + A_{s_j,q_{i,j}}) + a^{\mathbf{q}}_{i',j'}\right) & + & \\ -(w_{i'} + a^{\mathbf{w}}_{i'}) \cdot (r_{i'} + a^{\mathbf{r}}_{i'}) \cdot b^{\mathbf{sw}} - (w_{j'} + a^{\mathbf{w}}_{j'}) \cdot (s_{j'} + a^{\mathbf{s}}_{j'}) \cdot b^{\mathbf{rs}} & = & c. \end{array} \tag{10}$$

By assumption, we have that $w'_{i',j'} + a^{\mathbf{w}'}_{i',j'} - (w_{i'} + a^{\mathbf{w}}_{i'}) \cdot (w_{j'} + a^{\mathbf{w}}_{j'}) \neq 0$. Thus, Equation 10 has the form

$$c_1 r_{i'} s_{j'} + c_2 r_{i'} + c_3 s_{j'} + c_4 = 0 \tag{11}$$

where $c_1, c_2, c_3, c_4 \in \mathbb{F}$ and $c_1 \neq 0$. Thus the probability that Equation 11 above holds is $\frac{1}{|\mathbb{F}|} + \frac{|\mathbb{F}|-1}{|\mathbb{F}|^2}$. Therefore, we have proved that

$$\Pr_{\mathbf{r},\mathbf{s}} \left[\left(\langle \mathbf{w} + \mathbf{a}^{\mathbf{w}}, \mathbf{r} + \mathbf{a}^{\mathbf{r}} \rangle + b^{\mathbf{rw}} \right) \cdot \left(\langle \mathbf{w} + \mathbf{a}^{\mathbf{w}}, \mathbf{s} + \mathbf{a}^{\mathbf{s}} \rangle + b^{\mathbf{sw}} \right) \neq \langle \mathbf{w}' + \mathbf{a}^{\mathbf{w}'}, \mathbf{q} + \mathbf{a}^{\mathbf{q}} \rangle + b^{\mathbf{qw}'} \right]$$

is at least $\frac{(|\mathbb{F}|-1)^2}{|\mathbb{F}|^2}$. Combining this with Equation 7 above we obtain that

$$\begin{aligned}
&\Pr \left[(f_1 \neq 0 \vee f_2 \neq 0) \vee \left(v^{\mathbf{rw}} \cdot v^{\mathbf{sw}} \neq v^{\mathbf{qw}'} \right) \right] \qquad (12)\\
&\quad \geq \Pr \left[\begin{array}{c} \left(\langle \mathbf{w} + \mathbf{a}^{\mathbf{w}}, \mathbf{r} + \mathbf{a}^{\mathbf{r}} \rangle + b^{\mathbf{rw}} \right) \cdot \\ \left(\langle \mathbf{w} + \mathbf{a}^{\mathbf{w}}, \mathbf{s} + \mathbf{a}^{\mathbf{s}} \rangle + b^{\mathbf{sw}} \right) \\ \neq \\ \langle \mathbf{w}' + \mathbf{a}^{\mathbf{w}'}, \mathbf{q} + \mathbf{a}^{\mathbf{q}} \rangle + b^{\mathbf{qw}'} \end{array} \middle| \begin{array}{c} f_1 \neq 0 \vee f_2 \neq 0 \vee \\ \left(\begin{array}{c} v^{\mathbf{rw}} = \langle \mathbf{w} + \mathbf{a}^{\mathbf{w}}, \mathbf{r} + \mathbf{a}^{\mathbf{r}} \rangle + b^{\mathbf{rw}} \wedge \\ v^{\mathbf{sw}} = \langle \mathbf{w} + \mathbf{a}^{\mathbf{w}}, \mathbf{s} + \mathbf{a}^{\mathbf{s}} \rangle + b^{\mathbf{sw}} \wedge \\ v^{\mathbf{qw}'} = \langle \mathbf{w}' + \mathbf{a}^{\mathbf{w}'}, \mathbf{q} + \mathbf{a}^{\mathbf{q}} \rangle + b^{\mathbf{qw}'} \end{array} \right) \end{array} \right] \\
&\qquad \cdot \Pr \left[\begin{array}{c} f_1 \neq 0 \vee f_2 \neq 0 \vee \\ \left(\begin{array}{c} v^{\mathbf{rw}} = \langle \mathbf{w} + \mathbf{a}^{\mathbf{w}}, \mathbf{r} + \mathbf{a}^{\mathbf{r}} \rangle + b^{\mathbf{rw}} \wedge \\ v^{\mathbf{sw}} = \langle \mathbf{w} + \mathbf{a}^{\mathbf{w}}, \mathbf{s} + \mathbf{a}^{\mathbf{s}} \rangle + b^{\mathbf{sw}} \wedge \\ v^{\mathbf{qw}'} = \langle \mathbf{w}' + \mathbf{a}^{\mathbf{w}'}, \mathbf{q} + \mathbf{a}^{\mathbf{q}} \rangle + b^{\mathbf{qw}'} \end{array} \right) \end{array} \right] \\
&\quad \geq (1 - \epsilon')^2 \cdot \frac{(|\mathbb{F}| - 1)^2}{|\mathbb{F}|^2}
\end{aligned}$$

where the probability is over $\mathbf{r}, \mathbf{s}$ and the internal randomness of $C_{3\times t}$ and $C_{2\times t^2}$.

Recall that we assumed without loss of generality that A does not attack the entire sub-circuit inside $\widehat{C}$ whose inputs are $v^{\mathbf{rw}}, v^{\mathbf{sw}}, v^{\mathbf{qw}'}, v^{\mathbf{pw}}, v^{\mathbf{q}'\mathbf{w}'}, \mathbf{z}, \mathbf{u}, f_1, f_2, r_1, r_2, r_3, r_4$ and whose output is f in Step 12. Let $A_{f,\mathsf{output}}$ be A restricted on the wire carrying f to the output gate. We have that

$$\begin{aligned}
\Pr \left[\widetilde{C} \leftarrow A(\widehat{C}) : \widetilde{C}(\mathbf{x}) \in \mathsf{ERR} \right] &= \Pr \left[\widetilde{C} \leftarrow A(\widehat{C}) : f + A_{f,\mathsf{output}} \neq 0 \right] \\
&\geq \Pr_{r_1,r_2,r_3,r_4} \left[\widetilde{C} \leftarrow A(\widehat{C}) : \begin{array}{c} r_1(v^{\mathbf{rw}} \cdot v^{\mathbf{sw}} - v^{\mathbf{qw}'}) + \\ r_2(v^{\mathbf{pw}} - v^{\mathbf{q}'\mathbf{w}'} - \mathbf{z}^T \mathbf{u}) + r_3 f_1 + r_4 f_2 + A_{f,\mathsf{output}} \\ \neq 0 \end{array} \right] \\
&\geq \frac{|\mathbb{F}| - 1}{|\mathbb{F}|} \cdot \Pr \left[f_1 \neq 0 \vee f_2 \neq 0 \vee v^{\mathbf{rw}} \cdot v^{\mathbf{sw}} - v^{\mathbf{qw}'} \neq 0 \right] \\
&\geq (1 - \epsilon')^2 \cdot \frac{(|\mathbb{F}| - 1)^3}{|\mathbb{F}|^3}.
\end{aligned}$$

where the last transition follows from Equation 12. Hence the additive-attack correctness requirement of Definition 4.1 are fulfilled in this case.

2. In this case we have that

$$\mathbf{w}' + \mathbf{a}^{\mathbf{w}'} = (\mathbf{w} + \mathbf{a}^{\mathbf{w}}) \cdot (\mathbf{w} + \mathbf{a}^{\mathbf{w}})^T.$$

That is, for any $1 \leq i, j \leq t$ it holds that $w'_{i,j} + a^{\mathbf{w}'}_{i,j} = (w_i + a^{\mathbf{w}}_i) \cdot (w_j + a^{\mathbf{w}}_j)$. Notice that A causes the subcircuit computing $\mathbf{q}'$ to actually compute the following:

$$q'_{j,j'} = \sum_{i \in I_{j,j'}} p_i + A_{p_i, q'_{j,j'}}.$$

From Equation 6 above it holds that

$$\Pr\left[(f_1 = 0 \vee f_2 = 0) \vee \left(\begin{array}{c} v^{\mathbf{pw}} = \langle \mathbf{w} + \mathbf{a}^{\mathbf{w}}, \mathbf{p} + \mathbf{a}^{\mathbf{p}} \rangle + b^{\mathbf{pw}} \wedge \\ v^{\mathbf{q}'\mathbf{w}'} = \langle \mathbf{w}' + \mathbf{a}^{\mathbf{w}'}, \mathbf{q}' + \mathbf{a}^{\mathbf{q}'} \rangle + b'^{\mathbf{q}'\mathbf{w}'} \end{array} \right)\right] \geq (1-\epsilon')^2. \tag{13}$$

In the sequel we will show that the check $v^{\mathbf{pw}} - v^{\mathbf{q}'\mathbf{w}'} - \mathbf{z}^T\mathbf{u}$ in step 12 designed to check that the circuit C computed its gates correctly will fail with constant probability causing f to become random. We start by proving that the probability over the selection of $\mathbf{p} \in \mathbb{F}^t$ that $\langle \mathbf{w} + \mathbf{a}^{\mathbf{w}}, \mathbf{p} + \mathbf{a}^{\mathbf{p}} \rangle + b^{\mathbf{pw}} - \langle \mathbf{w}' + \mathbf{a}^{\mathbf{w}'}, \mathbf{q}' + \mathbf{a}^{\mathbf{q}'} \rangle - b^{\mathbf{q}'\mathbf{w}'} = \mathbf{z}^T\mathbf{u}$ is bounded by a universal constant. Indeed, this probability is bounded by the probability (over the selection of $\mathbf{p} \in \mathbb{F}^t$) that the following equation holds:

$$b^{\mathbf{pw}} + \sum_{i=1}^{t}(w_i + a_i^{\mathbf{w}}) \cdot (p_i + a_i^{\mathbf{p}}) - b^{\mathbf{q}'\mathbf{w}'} - \sum_{j=1}^{t^2}\left(w'_j + a_j^{\mathbf{w}'}\right) \cdot \left(q'_j + a_j^{\mathbf{q}'}\right) = \mathbf{z}^T\mathbf{u}. \tag{14}$$

We split the proof into three subcases again based on $w + a^{\mathbf{w}}$.

(a) There exists a gate inside C that is not computed correctly. That is, there exists a gate i' inside C whose inputs are the outputs of gate j' and j'' such that

$$w_{i'} + a_{i'}^{\mathbf{w}} \neq \left(\sum_{j' \in \mathsf{left}(i')} w_{j'} + a_{j'}^{\mathbf{w}}\right) \cdot \left(\sum_{j'' \in \mathsf{right}(i')} w_{j''} + a_{j''}^{\mathbf{w}}\right).$$

Equation 14 can be rewritten as

$$\begin{aligned} &(w_{i'} + a_{i'}^{\mathbf{w}}) \cdot (p_{i'} + a_{i'}^{\mathbf{p}}) - \sum_{\substack{j' \in \mathsf{left}(i') \\ j'' \in \mathsf{right}(i')}} \left(w'_{j',j''} + a_{j',j''}^{\mathbf{w}'}\right) \cdot \left(q'_{j',j''} + a_{j',j''}^{\mathbf{q}'}\right) \quad = \\ &b^{\mathbf{q}'\mathbf{w}'} - \left(\textstyle\sum_{i \neq i'}(w_i + a_i^{\mathbf{w}}) \cdot (p_i + a_i^{\mathbf{p}})\right) + \left(\sum_{\substack{i \notin \mathsf{left}(i') \\ j \notin \mathsf{right}(i')}} \left(w'_{i,j} + a'^{\mathbf{w}'}_{i,j}\right) \cdot \left(q'_{i,j} + a_{i,j}^{\mathbf{q}'}\right)\right) \quad + \\ &\mathbf{z}^T\mathbf{u} - b^{\mathbf{pw}}. \end{aligned} \tag{15}$$

We will show that for any selection of values to all $p_1, \ldots, p_t$ but $p_{i'}$ there exists a selection of $p_{i'}$ such that Equation 15 does not hold. Indeed, fix some values to all $p_1, \ldots, p_t$ but $p_{i'}$. Notice that the right hand side of Equation 15 does not depend on the value of $p_{i'}$. Denote by c the value of the right hand side of Equation 15. By assumption, it holds that

$$\begin{aligned} \sum_{\substack{j' \in \mathsf{left}(i') \\ j'' \in \mathsf{right}(i')}} w'_{j',j''} + a_{j',j''}^{\mathbf{w}'} &= \sum_{\substack{j' \in \mathsf{left}(i') \\ j'' \in \mathsf{right}(i')}} (w_{j'} + a_{j'}^{\mathbf{w}}) \cdot (w_{j''} + a_{j''}^{\mathbf{w}}) \\ &\neq w_{i'} + a_{i'}^{\mathbf{w}}. \end{aligned} \tag{16}$$

Thus, Equation 15 can be rewritten as

$$\left(w_{i'} + a_{i'}^{\mathbf{w}}\right) \cdot \left(p_{i'} + a_{i'}^{\mathbf{p}}\right) - \sum_{\substack{j' \in \mathsf{left}(i') \\ j'' \in \mathsf{right}(i')}} \left(w'_{j',j''} + a_{j',j''}^{\mathbf{w}'}\right) \cdot \left(p_{i'} + d\right) = c \tag{17}$$

where $d = A_{p_{i'},q'_{j',j''}} + \sum_{i \in I_{j,j'} \setminus \{i'\}} p_i + A_{p_i,q'_{j,j'}} + a_{j',j''}^{\mathbf{q}'}$.
Notice that Equation 17 above has the form

$$c_1 p_{i'} + c_2 = 0 \tag{18}$$

where $c_1, c_2 \in \mathbb{F}$ and $c_1 \neq 0$. Thus the probability that Equation 18 above holds is $1/|\mathbb{F}|$. Therefore, we have proved that

$$\Pr_{\mathbf{p}}\left[b^{\mathbf{pw}} + \langle \mathbf{w}+\mathbf{a}^{\mathbf{w}}, \mathbf{p}+\mathbf{a}^{\mathbf{p}}\rangle - b^{\mathbf{q}'\mathbf{w}'} - \langle \mathbf{w}'+\mathbf{a}^{\mathbf{w}'}, \mathbf{q}'+\mathbf{a}^{\mathbf{q}'}\rangle \neq \mathbf{z}^T\mathbf{u}\right] \geq \frac{|\mathbb{F}|-1}{|\mathbb{F}|}.$$

Combining with Equation 13 above we obtain that Combining this with Equation 7 above we obtain that

$$\begin{aligned}
&\Pr\left[(f_1 \neq 0 \vee f_2 \neq 0) \vee \left(v^{\mathbf{pw}} - v^{\mathbf{q}'\mathbf{w}'} \neq \mathbf{z}^T\mathbf{u}\right)\right] \qquad (19)\\
&\geq \Pr\left[\begin{array}{c} b^{\mathbf{pw}} + \langle \mathbf{w}+\mathbf{a}^{\mathbf{w}}, \mathbf{p}+\mathbf{a}^{\mathbf{p}}\rangle - \\ b^{\mathbf{q}'\mathbf{w}'} - \langle \mathbf{w}'+\mathbf{a}^{\mathbf{w}'}, \mathbf{q}'+\mathbf{a}^{\mathbf{q}'}\rangle \\ \neq \\ \mathbf{z}^T\mathbf{u} \end{array} \middle| \begin{array}{c} f_1 \neq 0 \vee f_2 \neq 0 \vee \\ \left(\begin{array}{c} v^{\mathbf{pw}} = \langle \mathbf{w}+\mathbf{a}^{\mathbf{w}}, \mathbf{p}+\mathbf{a}^{\mathbf{p}}\rangle + b^{\mathbf{pw}} \wedge \\ v^{\mathbf{q}'\mathbf{w}'} = \langle \mathbf{w}'+\mathbf{a}^{\mathbf{w}'}, \mathbf{q}'+\mathbf{a}^{\mathbf{q}'}\rangle + b'^{\mathbf{q}'\mathbf{w}'} \end{array}\right) \end{array}\right]\\
&\cdot \Pr\left[\begin{array}{c} f_1 \neq 0 \vee f_2 \neq 0 \vee \\ \left(\begin{array}{c} v^{\mathbf{pw}} = \langle \mathbf{w}+\mathbf{a}^{\mathbf{w}}, \mathbf{p}+\mathbf{a}^{\mathbf{p}}\rangle + b^{\mathbf{pw}} \wedge \\ v^{\mathbf{q}'\mathbf{w}'} = \langle \mathbf{w}'+\mathbf{a}^{\mathbf{w}'}, \mathbf{q}'+\mathbf{a}^{\mathbf{q}'}\rangle + b'^{\mathbf{q}'\mathbf{w}'} \end{array}\right) \end{array}\right]\\
&\geq (1-\epsilon')^2 \cdot \frac{(|\mathbb{F}|-1)}{|\mathbb{F}|}
\end{aligned}$$

where the probability is over $\mathbf{p}$ and the internal randomness of $C_{3\times t}$ and $C_{2\times t^2}$.
Recall that we assumed without loss of generality that A does not attack the entire sub-circuit inside $\widehat{C}$ whose inputs are $v^{\mathbf{rw}}, v^{\mathbf{sw}}, v^{\mathbf{qw}'}, v^{\mathbf{pw}}, v^{\mathbf{q}'\mathbf{w}'}, \mathbf{z}, \mathbf{u}, f_1, f_2, r_1, r_2, r_3, r_4$ and whose output is f in Step 12. Let $A_{f,\mathsf{output}}$ be A restricted on the wire carrying f to the output gate. We have that

$$\begin{aligned}
&\Pr\left[\widetilde{C} \leftarrow A(\widehat{C}) : \widetilde{C}(\mathbf{x}) \in \mathsf{ERR}\right] = \Pr\left[\widetilde{C} \leftarrow A(\widehat{C}) : f + A_{f,\mathsf{output}} \neq 0\right]\\
&\geq \Pr_{r_1,r_2,r_3,r_4}\left[\widetilde{C} \leftarrow A(\widehat{C}) : \begin{array}{c} r_1(v^{\mathbf{rw}} \cdot v^{\mathbf{sw}} - v^{\mathbf{qw}'}) + \\ r_2(v^{\mathbf{pw}} - v^{\mathbf{q}'\mathbf{w}'} - \mathbf{z}^T\mathbf{u}) + r_3 f_1 + r_4 f_2 + A_{f,\mathsf{output}} \\ \neq 0 \end{array}\right]\\
&\geq \frac{|\mathbb{F}|-1}{|\mathbb{F}|} \cdot \Pr\left[f_1 \neq 0 \vee f_2 \neq 0 \vee v^{\mathbf{pw}} - v^{\mathbf{q}'\mathbf{w}'} - \mathbf{z}^T\mathbf{u} \neq 0\right]\\
&\geq (1-\epsilon')^2 \cdot \frac{(|\mathbb{F}|-1)^2}{|\mathbb{F}|^2}
\end{aligned}$$

where the last transition follows from Equation 19. Hence the additive-attack correctness requirement of Definition 4.1 are fulfilled in this case.

(b) All the gates inside C are computed correctly and there exists an output wire that was corrupted. That is, for any wire i' inside C that is the output of a gate whose inputs are wires j' and j'' it holds that

$$(w_{i'} + a^{\mathbf{w}}_{i'}) = \sum_{\substack{j' \in \mathsf{left}(i') \\ j'' \in \mathsf{right}(i')}} (w_{j'} + a^{\mathbf{w}}_{j'}) \cdot (w_{j''} + a^{\mathbf{w}}_{j''})$$

and there exists an output k' of C such that $z_{k'} \neq w_{t-1+k'} + a^{\mathbf{w}}_{t-1+k'}$. Notice that since k' is an output of C, it holds that the values $\sum_{j=1}^{t^2}\left(w'_j + a_j^{\mathbf{w}'}\right) \cdot \left(q'_j + a_j^{\mathbf{q}'}\right)$ in Equation 14 do not depend on $p_{t-1+k'}$. Thus, Equation 14 can be rewritten as

$$\begin{array}{lcll}
(w_{t-1+k'} + a^{\mathbf{w}}_{t-1+k'}) \cdot \left(p_{t-1+k'} + a^{\mathbf{p}}_{t-1+k'}\right) & - & u_{t-1+k'} \cdot z_{t-1+k'} & = \\
-b^{\mathbf{pw}} - \left(\sum_{i \neq t-1+k'} (w_i + a^{\mathbf{w}}_i) \cdot (p_i + a^{\mathbf{p}}_i)\right) & + & \left(\sum_{i \neq k'} u_i z_i\right) & \qquad (20)\\
b^{\mathbf{q}'\mathbf{w}'} + \left(\sum_{j=1}^{t^2}\left(w'_j + a_j^{\mathbf{w}}\right) \cdot \left(q'_j + a_j^{\mathbf{q}'}\right)\right) & & &
\end{array}$$

Notice that the right hand side of Equation 20 does not depend on the value of $p_{t-1+k'}$ and that by assumption it holds that $z_{k'} \neq w_{t-1+k'} + a^{\mathbf{w}}_{t-1+k'}$. Thus Equation 20 has the form

$$c_1 p_{t-1+k'} + c_2 = 0 \tag{21}$$

where $c_1, c_2 \in \mathbb{F}$ and $c_1 \neq 0$. Therefore, we have proved that

$$\Pr_{\mathbf{p}}\left[b^{\mathbf{pw}} + \langle \mathbf{w} + \mathbf{a}^{\mathbf{w}}, \mathbf{p} + \mathbf{a}^{\mathbf{p}} \rangle - b^{\mathbf{q}'\mathbf{w}'} - \langle \mathbf{w}' + \mathbf{a}^{\mathbf{w}'}, \mathbf{q}' + \mathbf{a}^{\mathbf{q}'} \rangle \neq \mathbf{z}^T \mathbf{u}\right] \geq \frac{|\mathbb{F}| - 1}{|\mathbb{F}|}.$$

Combining with Equation 13 above we obtain that

$$\begin{aligned}
&\Pr\left[(f_1 \neq 0 \vee f_2 \neq 0) \vee \left(v^{\mathbf{pw}} - v^{\mathbf{q}'\mathbf{w}'} \neq \mathbf{z}^T \mathbf{u}\right)\right] \qquad (22)\\
&\geq \Pr\left[\begin{array}{c} b^{\mathbf{pw}} + \langle \mathbf{w} + \mathbf{a}^{\mathbf{w}}, \mathbf{p} + \mathbf{a}^{\mathbf{p}} \rangle - \\ b^{\mathbf{q}'\mathbf{w}'} - \langle \mathbf{w}' + \mathbf{a}^{\mathbf{w}'}, \mathbf{q}' + \mathbf{a}^{\mathbf{q}'} \rangle \\ \neq \\ \mathbf{z}^T \mathbf{u} \end{array} \middle| \begin{array}{c} f_1 \neq 0 \vee f_2 \neq 0 \vee \\ \left(\begin{array}{c} v^{\mathbf{pw}} = \langle \mathbf{w} + \mathbf{a}^{\mathbf{w}}, \mathbf{p} + \mathbf{a}^{\mathbf{p}} \rangle + b^{\mathbf{pw}} \wedge \\ v^{\mathbf{q}'\mathbf{w}'} = \langle \mathbf{w}' + \mathbf{a}^{\mathbf{w}'}, \mathbf{q}' + \mathbf{a}^{\mathbf{q}'} \rangle + b'^{\mathbf{q}'\mathbf{w}'} \end{array}\right) \end{array}\right]\\
&\cdot \Pr\left[\begin{array}{c} f_1 \neq 0 \vee f_2 \neq 0 \vee \\ \left(\begin{array}{c} v^{\mathbf{pw}} = \langle \mathbf{w} + \mathbf{a}^{\mathbf{w}}, \mathbf{p} + \mathbf{a}^{\mathbf{p}} \rangle + b^{\mathbf{pw}} \wedge \\ v^{\mathbf{q}'\mathbf{w}'} = \langle \mathbf{w}' + \mathbf{a}^{\mathbf{w}'}, \mathbf{q}' + \mathbf{a}^{\mathbf{q}'} \rangle + b'^{\mathbf{q}'\mathbf{w}'} \end{array}\right) \end{array}\right]\\
&\geq (1 - \epsilon')^2 \cdot \frac{(|\mathbb{F}| - 1)}{|\mathbb{F}|}
\end{aligned}$$

where the probability is over $\mathbf{p}$ and the internal randomness of $C_{3\times t}$ and $C_{2\times t^2}$.
Recall that we assumed without loss of generality that A does not attack the entire sub-circuit inside $\widehat{C}$ whose inputs are $v^{\mathbf{rw}}, v^{\mathbf{sw}}, v^{\mathbf{qw}'}, v^{\mathbf{pw}}, v^{\mathbf{q}'\mathbf{w}'}, \mathbf{z}, \mathbf{u}, f_1, f_2, r_1, r_2, r_3, r_4$ and whose output is f in Step 12. Let $A_{f,\mathsf{output}}$ be A restricted on the wire carrying f to the output gate. We have that

$$\begin{aligned}
&\Pr\left[\widetilde{C} \leftarrow A(\widehat{C}) : \widetilde{C}(\mathbf{x}) \in \mathsf{ERR}\right] = \Pr\left[\widetilde{C} \leftarrow A(\widehat{C}) : f + A_{f,\mathsf{output}} \neq 0\right]\\
&\geq \Pr_{r_1,r_2,r_3,r_4}\left[\widetilde{C} \leftarrow A(\widehat{C}) : \begin{array}{c} r_1(v^{\mathbf{rw}} \cdot v^{\mathbf{sw}} - v^{\mathbf{qw}'}) + \\ r_2(v^{\mathbf{pw}} - v^{\mathbf{q}'\mathbf{w}'} - \mathbf{z}^T\mathbf{u}) + r_3 f_1 + r_4 f_2 + A_{f,\mathsf{output}} \\ \neq 0 \end{array}\right]\\
&\geq \frac{|\mathbb{F}| - 1}{|\mathbb{F}|} \cdot \Pr\left[f_1 \neq 0 \vee f_2 \neq 0 \vee v^{\mathbf{pw}} - v^{\mathbf{q}'\mathbf{w}'} - \mathbf{z}^T\mathbf{u} \neq 0\right]\\
&\geq (1 - \epsilon')^2 \cdot \frac{(|\mathbb{F}| - 1)^2}{|\mathbb{F}|^2}
\end{aligned}$$

where the last transition follows from Equation 22. Hence the additive-attack correctness requirement of Definition 4.1 are fulfilled in this case.

(c) The circuit C was computed correctly. That is, for any wire i' inside C that is the output of a gate whose inputs are wires j' and j'' it holds that

$$(w_{i'} + a^{\mathbf{w}}_{i'}) = \sum_{\substack{j' \in \mathsf{left}(i') \\ j'' \in \mathsf{right}(i')}} (w_{j'} + a^{\mathbf{w}}_{j'}) \cdot (w_{j''} + a^{\mathbf{w}}_{j''})$$

and for any output wire k' it holds that $z_{k'} = w_{t-1+k'} + a^{\mathbf{w}}_{t-1+k'}$. Then we have obtained that the vector $\mathbf{w} + \mathbf{a}^{\mathbf{w}}$ is a valid vector of the intermediate values of the computation of the circuit C on input x. Recall that $\mathbf{a}^{\mathsf{in}} = \mathbf{a}^{\mathbf{x}} + \mathbf{a}^{\mathbf{w},\mathsf{inp}}$ where $\mathbf{a}^{\mathbf{x}}$ is A restricted to the wires

carrying the input $\mathbf{x}$ inside the vector $\mathbf{w}$ in C and $\mathbf{a}^{\mathbf{w},\mathsf{inp}} \in \mathbb{F}^n$ is a vector of entries in $\mathbf{a}^{\mathbf{w}}$ corresponding to the input wires of C. In addition, recall that $\mathbf{a}^{\mathsf{out}}$ is the restriction of A on the output wires of C. Thus, we have obtained that $z = C(\mathbf{x} + \mathbf{a}^{\mathsf{in}}) + \mathbf{a}^{\mathsf{out}}$. Recall that $\mathsf{ERR} = \{(z', z'') : z' \in \mathbb{F} \setminus \{0\}, z'' \in \mathbb{F}^k\}$. Thus,

$$\Pr\left[\widetilde{C} \leftarrow A(\widehat{C}) : \widetilde{C}(\mathbf{x}) \in \mathsf{ERR} \cup \{(0, C(\mathbf{x} + \mathbf{a}^{\mathsf{in}}) + \mathbf{a}^{\mathsf{out}})\}\right] = 1.$$

Hence the additive-attack correctness requirement of Definition 4.1 are fulfilled in this case.

■

4.2 Improving efficiency and correctness using a decoder

In this section we achieve one of the goals outlined in Section 1.2.1. That is, we present a transformation that will have all the properties outlined in Definition 1.2. Notice that the results of this section are not comparable with the results presented in Section 4.1 above for several reasons. First, the soundness guarantee of the results presented in Section 4.1 relies on the size of the underlying field over which the circuits are constructed. While the construction in Section 4.1 does have some soundness over small fields, we do not have a way to amplify this soundness (without increasing the field size). The second issue is related to the overhead of the main construction presented in Section 4.1.3 which is quadratic in the size of the circuit.

We achieve these goals in a model where one is allowed a *decoder* that cannot be tampered with in order to verify the results of the tampered circuit; this is captured by the notion of *additive-attack correctness with decoder* (Definition 1.2). In Section 4.2.1 we present a method for amplification of any correct construction without increasing the size of the underlying field. Then, in Section 4.2.2, we present a construction with linear overhead that achieves additive-attack correctness with decoder.

4.2.1 Correctness amplification using hashing

In Construction 4.3 we presented an $|\mathbb{F}|^{-\Omega(1)}$-correct construction for arbitrary circuits. However, for this construction to work one must require that the size of the underlying field $\mathbb{F}$ on which the circuit computes will be relatively large. If we fix $\mathbb{F}$ to be the binary field for instance we obtain a circuit that is correct with very small probability. In this section we achieve two different goals. First, we present a construction that will amplify the probability that the circuit detects an attack on it. Second, we would like that for any additive attack A, the input attack corresponding to it will not depend on the way A attacks the outputs of $\widehat{C}$. Our construction will use replication in order to check that all of the circuits computed correctly the same output. However, such an approach requires a *tamper-proof* output decoder to verify consistency between all instances of the circuit. Formally, for any circuit C and positive integer σ we would like to construct a $2^{-\sigma}$-strongly correct implementation with a decoder of C as defined in Definition 4.2 below.

Definition 4.2. *Let $C : \mathbb{F}^n \to \mathbb{F}^k$ be an arithmetic circuit. We say that a pair of circuits $(\widehat{C}, D)$ are an ϵ-strongly correct implementation of C with a decoder if the following holds:*

- *$(\widehat{C}, D)$ are an ϵ-correct implementation of C.*
- *For any additive attack A, the vector $\mathbf{a}^{\mathsf{in}} \in \mathbb{F}^n$ from the additive-attack correctness property of $(\widehat{C}, D)$ doesn't depend on the values that A adds to the output wires of $\widehat{C}$.*

Why naive amplification does not work. As usual, we would like to amplify the correctness of an additively correct scheme using repetitions. That is, given an ϵ-additively correct circuit, we expect

that by running the construction σ times and accepting only if the outputs are consistent we will obtain a construction with better soundness. Unfortunately, this is not true since the adversary is allowed to attack each instance in the repetition differently. Such an attack strategy might lead to different ideal attacks on the inputs of each instance. In this case, no amplification will be achieved.

As a counter example let $\mathbb{F} = \{0,1\}$ be the binary field, n, σ be a positive integers such that $|\mathbb{F}|^n > \sigma$. Consider the circuit $C : \mathbb{F}^n \to \mathbb{F}$ defined as follows:

$$C(\mathbf{x}) = \begin{cases} 0 & \text{if } \mathbf{x} = 0, \\ 1 & \text{otherwise.} \end{cases}$$

In addition, for any $\mathbf{a} \in \mathbb{F}^n$ let $\widetilde{C}_{\mathbf{a}} : \mathbb{F}^n \to \mathbb{F}$ be a randomized circuit defined as follows:

$$\widetilde{C}_{\mathbf{a}}(\mathbf{x}) = \begin{cases} (0, r) & \text{if } \mathbf{x} = \mathbf{a}, \\ (0, 1) & \text{otherwise} \end{cases}$$

where r is a random field element. We define the class $\widetilde{\mathcal{C}}$ to be the set of all circuits $\widetilde{C}_{\mathbf{a}}$ as defined above for all possible $\mathbf{a} \in \mathbb{F}^n$. We note that for any $\widetilde{C}_{\mathbf{a}} \in \widetilde{\mathcal{C}}$ it holds that for any $\mathbf{x} \in \mathbb{F}^n$

$$\Pr\left[\widetilde{C}_{\mathbf{a}}(\mathbf{x}) \notin \{(0, C(\mathbf{x} - \mathbf{a}))\} \cup \mathsf{ERR}\right] \leq \frac{1}{2}.$$

However, let $\{\widetilde{C}_{\mathbf{a}_1}, \ldots, \widetilde{C}_{\mathbf{a}_\sigma}\}$ be a set of circuits such that for any i, j it holds that $\mathbf{a}_i \neq \mathbf{a}_j$. We claim that for any $\mathbf{a} \in \mathbb{F}^n$ there exists an $\mathbf{x} \in \mathbb{F}^n$ such that

$$\Pr\left[\left(\forall i, j : \widetilde{C}_{\mathbf{a}_i}(\mathbf{x}) = \widetilde{C}_{\mathbf{a}_j}(\mathbf{x})\right) \wedge \widetilde{C}_{\mathbf{a}_1}(\mathbf{x}) \notin \mathsf{ERR} \cup \{(0, C(\mathbf{x} - \mathbf{a}))\}\right] \geq 1/2.$$

Indeed, let $\mathbf{a} \in \mathbb{F}^n$ and let $\mathbf{x} = \mathbf{a}$ notice that $C(\mathbf{x} - \mathbf{a}) = C(0) = 0$ and for all but at most one of $C_{\mathbf{a}_i}$'s in the above it holds that $C_{\mathbf{a}_i}(\mathbf{x}) = 1$. Thus, we have that for $\mathbf{x} = \mathbf{a}$ it holds that

$$\Pr\left[\left(\forall i, j : \widetilde{C}_{\mathbf{a}_i}(\mathbf{x}) = \widetilde{C}_{\mathbf{a}_j}(\mathbf{x})\right) \wedge \widetilde{C}_{\mathbf{a}_1}(\mathbf{x}) \notin \mathsf{ERR} \cup \{(0, C(\mathbf{x} - \mathbf{a}))\}\right] \geq 1/2.$$

Thus no amplification is achieved.

Correctness amplification using hashing. Intuitively, the problem presented in the previous section originated from the fact that every instance of the repeated circuit could have worked on a different input. In order to prevent this from happening we would like to make sure that all of the circuits worked on the same input. One obvious way to achieve this to modify the circuit before the replication, to include its inputs inside its outputs (in additional to its previous outputs) and then replicate the modified circuit. Thus, if the inputs where to differ, the decoder would be able to pick it up. However, this trivial solution has an unwanted property where the size of the decoder will be polynomial in the input length of the original circuit. In order to construct a pair of circuits $(\widehat{C}, D)$ that is a $2^{-\sigma}$-correct implementation of a circuit $C : \mathbb{F}^n \to \mathbb{F}^k$ such that the size of D is $\mathrm{poly}(\sigma, k, \log|\mathbb{F}|, \log n)$ we will use almost universal hash functions. Our strategy will be as follows. We start from a circuit C and modify it to a circuit C' that will also output a *hash digest* of its input instead of the input itself in addition to the regular output of C. Next, we take an arbitrary ϵ-correct implementation $\widehat{C}'$ of C and replicate it $O(\sigma)$ times. In this way we obtain a circuit that is $\exp(-\sigma)$-correct implementation of C. Since the size of the hash digest has length that is $\mathrm{poly}(\sigma, \log|\mathbb{F}|, \log n)$ we obtain the desired properties. First, we define what a almost universal has function family is. Formally,

Definition 4.3. *Let $\mathbb{F}$ be a finite field. A family of functions $\mathcal{H} = \{h_n : \mathbb{F}^n \times \mathbb{F}^{t(n)} \to \mathbb{F}^{l(n)}\}$ is a family of $(n, t(n), l(n), \epsilon(n))$-almost-universal-hash-functions over $\mathbb{F}$ if for any $n \in \mathbb{N}$, $\mathbf{x}, \mathbf{y} \in \mathbb{F}^n$ such that $\mathbf{x} \neq \mathbf{y}$ it holds that*

$$\Pr_{\mathbf{r} \in \mathbb{F}^{t(n)}}[h(\mathbf{x}, \mathbf{r}) = h(\mathbf{y}, \mathbf{r})] \leq \epsilon(n).$$

The following theorem states that there exist almost universal hash function families with short enough output over any finite field $\mathbb{F}$.

Theorem 4.2 (Cf. [WC79, CW79, WC81, NN93])**.** *Let $\mathbb{F}$ be a finite field. There exists a $(n, \Theta(\log n+\sigma), 2\sigma, 2^{-\sigma})$-almost-universal-hash function family over $\mathbb{F}$.*

We are now ready to define the transformation that takes a circuit C into a circuit C_{Hash} such that C_{Hash} outputs the same output as C and a hash digest of the input to C. In addition, since we would like to prevent attacks on the outputs of C we would like C_{Hash} to encode its outputs with an additively secure code.

Construction 4.4. *Let $\mathbb{F}$ be a finite field, $\mathcal{H} = \{h_n\}_{n\in\mathbb{N}}$ be a family of $(n, t(n), l(n), \epsilon(n))$-almost-universal-hash-functions over $\mathbb{F}$ and let and $C : \mathbb{F}^n \to \mathbb{F}^k$ be a circuit. In addition, let $(\mathsf{Enc}, \mathsf{Dec})$ be a $(k + l(n), q(n), \epsilon_{\mathsf{AMD}})$-AMD code. We define the circuit $C_{\mathsf{Hash}} : \mathbb{F}^n \times \mathbb{F}^{t(n)} \to \mathbb{F}^{q(n)}$ with respect to $(C, \mathcal{H}, \mathsf{Enc}, \mathsf{Dec})$ as follows: C_{Hash} on the input $(\mathbf{x}, \mathsf{hk})$ outputs $\mathsf{Enc}(C(\mathbf{x})||h_n(\mathbf{x}, \mathsf{hk}))$.*

We are now ready to define our main construction for correctness amplification.

Construction 4.5 (Correctness amplification for arbitrary circuits)**.** *Let $\mathbb{F}$ be a finite field, σ be a positive integer, $C : \mathbb{F}^n \to \mathbb{F}^k$ be a circuit, $\mathcal{H}$ be a $(n, t(n, \sigma), 2\sigma, 2^{-\sigma})$-almost-universal-hash function family were $t(n, \sigma) \in \Theta(\log n + \sigma)$, $(\mathsf{Enc}, \mathsf{Dec})$ be a $(k + l(n) + t(n, \sigma), q(n), \epsilon_{\mathsf{AMD}})$-AMD code and let C_{Hash} be the circuit constructed in Construction 4.4 with respect to $(C, \mathcal{H}, \mathsf{Enc}, \mathsf{Dec})$. In addition, let $\widehat{C}_{\mathsf{Hash}}$ be an ϵ-correct implementation of C_{Hash}. Consider the circuits $(\widehat{C}, D)$ that are defined as follows:*

- *$\widehat{C}$ is composed of 2σ copies of $\widehat{C}_{\mathsf{Hash}}$ numbered from $\widehat{C}_{\mathsf{Hash},1}$ to $\widehat{C}_{\mathsf{Hash},2\sigma}$. $\widehat{C}$ on input $\mathbf{x}$ samples a vector $\mathbf{r}$ uniformly at random from $\mathbb{F}^{t(n,\sigma)}$ and outputs*

$$(\mathbf{z}_1, \ldots, \mathbf{z}_{2\sigma}) \leftarrow (\widehat{C}_{\mathsf{Hash},1}(\mathbf{x}, \mathbf{r}), \ldots, \widehat{C}_{\mathsf{Hash},2\sigma}(\mathbf{x}, \mathbf{r})).$$

- *D on input $(\mathbf{z}_1, \ldots, \mathbf{z}_{2\sigma})$ preforms the following computations:*
  - *For all $1 \le i \le 2\sigma$ compute $(b_i, \mathbf{z}'_i) \leftarrow \mathsf{Dec}(\mathbf{z}_i)$. If there exists i such that $b_i \neq 0$ then output $(1, 0^k)$.*
  - *If there exists a pair (i, j) such that $\mathbf{z}'_i \neq \mathbf{z}'_j$ then output $(1, 0^k)$.*
  - *Otherwise, output $(0, \mathbf{z})$ were z is the first k field elements of $\mathbf{z}'_1$.*

We now claim that Construction 4.5 indeed allows us to obtain for any circuit C and security parameter σ a $2^{-\sigma}$-correct implementation of C with a decoder such that the size of the decoder is polynomial in σ and the output size of C.

Theorem 4.3. *For any circuit $C : \mathbb{F}^n \to \mathbb{F}^k$ the circuits $(\widehat{C}, D)$ as obtained via Construction 4.5 above with respect to C are an $c^{-\sigma}$-strongly correct implementation of C with a decoder for some universal $c \in \mathbb{N}$. Moreover, $|\widehat{C}| = poly(\sigma, |C|)$ and $|D| = poly(\sigma, k, \log n)$.*

Proof. The completeness and complexity properties easily follow from the construction of $\widehat{C}$. To show the additive-attack correctness property, consider any additive attack A on $\widehat{C}$ and let $A_1, \ldots, A_{2\sigma}$ be A restricted to $\widehat{C}_{\mathsf{Hash},1}, \ldots, \widehat{C}_{\mathsf{Hash},2\sigma}$ respectively. By the additive-attack correctness property of $\widehat{C}_{\mathsf{Hash},1}, \ldots, \widehat{C}_{\mathsf{Hash},2\sigma}$ it holds that for any $1 \le i \le 2\sigma$ there exists $\mathbf{a}_i^{\mathsf{in}}, \mathbf{a}_i^{\mathsf{out}}$ such that for any $\mathbf{x} \in \mathbb{F}^n$, $\mathbf{r} \in \mathbb{F}^{t(n)}$

$$\Pr\left[\widetilde{C}_{\mathsf{Hash},\mathsf{i}} \leftarrow A_i(\widehat{C}_{\mathsf{Hash},\mathsf{i}}) : \widetilde{C}_{\mathsf{Hash},\mathsf{i}}(\mathbf{x}, \mathbf{r}) \notin \ \mathsf{ERR} \cup \{(0, \widetilde{C}_{\mathsf{Hash},\mathsf{i}}((\mathbf{x}, \mathbf{r}) + \mathbf{a}_i^{\mathsf{in}}) + \mathbf{a}_i^{\mathsf{out}})\}\ \right] \le \epsilon.$$

We split the proof into three cases based on the A:

1. There exists a set $U \subseteq \{1,\ldots,2\sigma\}$ such that $|U| \geq \sigma$ and for all $i \in U$ it holds that $\mathbf{a}_i^{\mathsf{out}} \neq 0$. Then, by the definition of D and the additive robustness of $(\mathsf{Enc}, \mathsf{Dec})$ we have that for any $\mathbf{x} \in \mathbb{F}^n$

$$\Pr\left[\widetilde{C} \leftarrow A(\widehat{C}) : D(\widetilde{C}(\mathbf{x})) \notin \mathsf{ERR}\right] \quad \leq \quad (1-(1-\epsilon)(1-\epsilon_{\mathsf{AMD}}))^{\sigma}.$$

2. For any set $U \subseteq \{1,\ldots,2\sigma\}$ such that $|U| \geq \sigma$, and for any $i \in U$ is holds that $\mathbf{a}_i^{\mathsf{out}} = 0$. Let U be such a set. We split the proof into two cases based on the number of distinct $\mathbf{a}_i^{\mathsf{in}}$'s such that $i \in U$.

   - There exists $\mathbf{a}$ such that $|\{i \in U : \mathbf{a}_i^{\mathsf{in}} = \mathbf{a}\}| \geq \sigma/2$. Thus we obtain that for any $\mathbf{x} \in \mathbb{F}^n$ it holds that

   $$\Pr\left[\widetilde{C} \leftarrow A(\widehat{C}) : D(\widetilde{C}(\mathbf{x})) \notin \mathsf{ERR} \cup \{(0, C(\mathbf{x}+\mathbf{a}))\}\right] \quad \leq \quad \epsilon^{\sigma/2}.$$

   - For any $\mathbf{a}$ it holds that $|\{i \in U : \mathbf{a}_i^{\mathsf{in}} = \mathbf{a}\}| < \sigma/2$. In this case there exists two sets $T, T' \subseteq U$ such that $|T|, |T'| \geq \sigma/4$ and $\{a_i : i \in T\} \cap \{a_i : i \in T'\} = \emptyset$. Next, by the definition of C_{Hash} we have that for any $\mathbf{x}, \mathbf{a}_1^{\mathsf{in}}, \mathbf{a}_2^{\mathsf{in}} \in \mathbb{F}^n$ such that $\mathbf{a}_1^{\mathsf{in}} \neq \mathbf{a}_2^{\mathsf{in}}$

   $$\Pr_{\mathbf{r}}\left[\ C_{\mathsf{Hash}}((\mathbf{x},\mathbf{r}) + \mathbf{a}_1^{\mathsf{in}}) \neq C_{\mathsf{Hash}}((\mathbf{x},\mathbf{r}) + \mathbf{a}_2^{\mathsf{in}})\ \right] \quad \geq \quad 1 - 2^{-\sigma}.$$

   Next, for any $\mathbf{x} \in \mathbb{F}^n$ it holds that

   $$\Pr\left[\begin{array}{c} \widetilde{C} \leftarrow A(\widehat{C}) \\ (\mathbf{z}_1,\ldots,\mathbf{z}_{2\sigma}) \leftarrow \widetilde{C}(\mathbf{x}) \end{array} : \begin{array}{c} \text{There exists } i \in T, j \in T' \text{ such that} \\ z_i \in \mathsf{IDEAL}_i \text{ and } z_j \in \mathsf{IDEAL}_j \end{array}\right] \quad \geq \quad (1-\epsilon^{\sigma/4})^2$$

   where $\mathsf{IDEAL}_i = \mathsf{ERR} \cup \{(0, \widehat{C}_{\mathsf{Hash,i}}((\mathbf{x},\mathbf{r}) + \mathbf{a}_i^{\mathsf{in}}))\}$.
   Next, by the definition of D, as soon as there exists $\mathbf{z}_i, \mathbf{z}_j$ such that $\mathbf{z}_i \neq \mathbf{z}_j$ we have that D outputs $(1, 0^k)$. Thus, we have that for any $\mathbf{x} \in \mathbb{F}^n$ it holds that

   $$\Pr\left[\widetilde{C} \leftarrow A(\widehat{C}) : D(\widetilde{C}(\mathbf{x})) \in \mathsf{ERR}\right] \geq$$

   $$\left(1-2^{-\sigma}\right) \cdot \Pr\left[\begin{array}{c} \widetilde{C} \leftarrow A(\widehat{C}) \\ (\mathbf{z}_1,\ldots,\mathbf{z}_{2\sigma}) \leftarrow \widetilde{C}(\mathbf{x}) \end{array} : \begin{array}{c} \text{There exists } i \in T, j \in T' \text{ such that} \\ z_i \in \mathsf{IDEAL}_i \text{ and } z_j \in \mathsf{IDEAL}_j \end{array}\right] \geq$$

   $$\left(1-2^{-\sigma}\right)\left(1-\epsilon^{\sigma/4}\right)$$

   where $\mathsf{IDEAL}_i = \mathsf{ERR} \cup \{(0, \widehat{C}_{\mathsf{Hash,i}}((\mathbf{x},\mathbf{r}) + \mathbf{a}_i^{\mathsf{in}}))\}$.

■

4.2.2 Additive-attack correctness with linear overhead

In Section 4.2.1 we presented an additively correct construction for arbitrary circuits with arbitrary high correctness. However, this construction did not preserve the communication graph of the circuit and produced a circuit with quadratic size of the original circuit. That is, we would like to obtain a transformation that on input a circuit $C : \mathbb{F}^n \to \mathbb{F}^k$ constructs the circuits $(\widehat{C}, D)$ such that two components G_u and G_v in $\widehat{C}$ corresponding to gates u and v in C are connected in $\widehat{C}$ if and only if u and v are connected in C. Intuitively the main idea is a follows. We will go over C from the input layer to the output layer, in each layer we will run the transformation in Construction 4.5 on every gate and obtain a additive-attack secure version of the gate together with a decoder. Next, we will consider the decoders as part of the next layer and will run the transformation in Construction 4.5 on it. After going through the circuit from the input layer to the output layer, the decoders for the output layer will be part of the final decoder. Formally, we would like to obtain a transformation that has the properties described in Definition 1.2. Consider Construction 4.6 below.

Construction 4.6. *Let $C : \mathbb{F}^n \to \mathbb{F}^k$ be a circuit containing only bilinear gates let σ be a positive integer. Consider the circuits $(\widehat{C}, D)$ defined as follows*

1. *Set $\widehat{C}$ to be a copy of C.*
2. *For any gate v with t left inputs and t' right inputs define the circuit F_v that performs the following*
   (a) *For any wire $w \in \mathsf{left}(v)$ if w is the output of some gate u then compute $(b_w, x_w) \leftarrow D_u(w)$ where D_u is obtained in step 3 below. Otherwise, if w is connected to the input set $x_w \leftarrow w$.*
   (b) *For any wire $w \in \mathsf{right}(v)$ if w is the output of some gate u then compute $(b_w, y_w) \leftarrow D_u(w)$ where D_u is obtained in step 3 below. Otherwise, if w is connected to the input set $y_w \leftarrow w$.*
   (c) *Compute $z \leftarrow (\sum_w x_w) \cdot (\sum_w y_w)$*
   (d) *If there exists w such that $b_w \neq 0$ output $(1, 0)$. Otherwise, output $(0, z)$.*
3. *Let $(\widehat{F}_v, D_v)$ be an $2^{-\sigma}$-strongly correct implementation of F_u with a decoder. Replace v with $\widehat{F}_v$ in $\widehat{C}$ and save D_v for later use in steps 2a and 2b above.*
4. *Let $v_1, \ldots, v_k$ be the gates inside C that are connected to the output such that v_i is connected to the ith output. The circuit D performs the following:*
   (a) *For any $1 \leq i \leq k$, D computes the output $(b_i, (b'_i, z_i))$ of D_{v_i} where the inputs of D_{v_i} are connected to the outputs of F_{v_i}.*
   (b) *If there exists i such that $b_i \neq 0$ or $b'_i \neq 0$ then output $(1, 0^k)$. Otherwise, output $(0, z_1 \ldots z_k)$.*

Theorem 4.4. *For any circuit C containing only bilinear gates and for any positive integer σ the size of D as defined in Construction 4.6 above is poly(σ, k).*

Proof. By the definition of correctness with a decoder, the size of D_{v_i} is polynomial in the output size of F_{v_i} and σ. Notice that the output size of F_{v_i} can be bounded by 2. ■

We now claim that Construction 4.6 indeed allows us to obtain for any circuit C and security parameter σ a $2^{-\sigma}$-correct implementation $\widehat{C}$ of C with a decoder such that the size of $\widehat{C}$ is $|C| \cdot \mathrm{poly}(\sigma)$ and the size of the decoder is polynomial in σ and the output size of C.

Theorem 4.5. *For any field $\mathbb{F}$, an arithmetic circuit $C : \mathbb{F}^n \to \mathbb{F}^k$ containing only bilinear gates and a positive integer σ the circuits $(\widehat{C}, D)$ as obtained via Construction 4.6 above are an $2^{-\sigma} \cdot |C|$-correct implementation of C where $|\widehat{C}| = |C| \cdot$ poly(σ) and $|D| =$ poly(k, σ).*

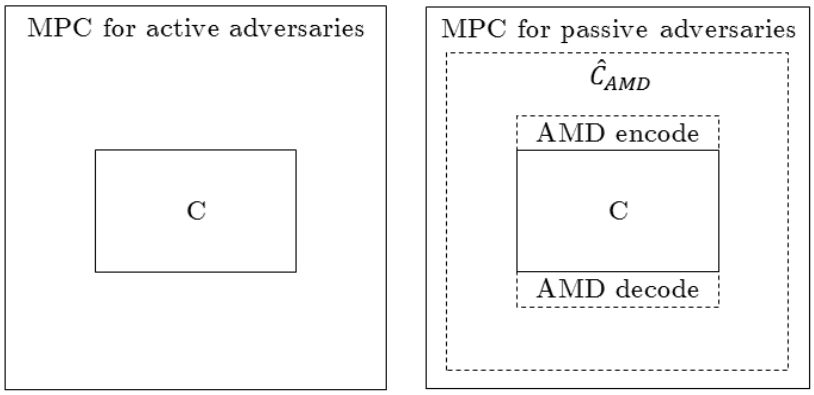
Proof. The completeness property easily follows from the definition of $\widehat{C}$. In addition, the complexity property easily follows from Theorem 4.4. Let A be an additive attack and let $\mathbf{x} \in \mathbb{F}^n$. In addition, for any gate v inside C let A_v be A restricted to $\widehat{F}_v$ inside $\widehat{C}$. We look at a highest $\widetilde{F}_v \leftarrow A_v(\widehat{F}_v)$ in $\widetilde{C} \leftarrow A(\widehat{C})$ that is attacked. By the additive-attack security property of $\widetilde{F}_v$ we obtain that there exists $\mathbf{a}_v^{\mathsf{in}}$ such that for any $\mathbf{x}$ it holds that

$$\Pr[D_v(\widetilde{F}_v(\mathbf{x})) \notin \mathsf{ERR} \cup \{(0, F_v(\mathbf{x} + \mathbf{a}_v^{\mathsf{in}}))\}] \leq 2^{-\sigma}.$$

Since every $\widehat{F}_v$ is a strongly correct implementation of F_v we have that $\mathbf{a}_v^{\mathsf{in}}$ does not depend on A_v restricted to the output wires of $\widehat{F}_v$. Thus, we can apply the above argument on all the $\widehat{F}_v$'s inside $\widehat{C}$ treating each attack on the inputs of $\widetilde{F}_v$ as an attack on the outputs of all the $\widetilde{F}_u$'s connected to $\widetilde{F}_v$ even in case of a fanout where multiple $\widetilde{F}_v$'s use the output of a single $\widetilde{F}_u$ inside $\widetilde{C}$. Thus, we have that there exists $\mathbf{a}^{\mathsf{in}} \in \mathbb{F}^n$ such that for all $\mathbf{x} \in \mathbb{F}^n$

$$\Pr\left[D(\widetilde{C}(\mathbf{x})) \notin \mathsf{ERR} \cup \{(0, C(\mathbf{x} + \mathbf{a}^{\mathsf{in}}))\}\right] \;\leq\; |C| \cdot \frac{1}{2^\sigma}.$$

■



(a) Traditional methodology (b) Our methodology

Figure 2: A graphical comparison between two methodologies for designing MPC protocols that are secure against active adversaries.

5 Secure MPC protocols from AMD circuits

In this section we construct multiparty computation protocols secure against active adversaries. Revisiting many protocols designed for MPC in the presence of a *passive* adversary, such as the GMW protocol [GMW87], the BGW protocol [BGW88] and more recently the DN protocol [DN07], we notice that when invoking these protocols in the presence of a *active* adversary, any deviation performed by the adversary during the execution of the protocol in fact corresponds to an additive attack on the underlying circuit. Thus, instead of designing an MPC protocol for computing an arithmetic circuit C that directly handles the presence on an active adversary, we use a protocol that is secure in the presence of *passive* adversaries but apply it to a modified circuit $\widehat{C}^{\mathsf{AMD}}$ which, in addition to computing the functionality computed by C, is also responsible for aborting the computation in case the adversary deviates from the protocol in a way that changes the protocol's outputs. The circuit $\widehat{C}^{\mathsf{AMD}}$ will be an additive-attack secure version of C^{AMD} which is the same as C with the exception that it gets its inputs and computes its outputs encoded in some AMD code. Figure 2 is a graphical representation of both the traditional methodology and our methodology for designing MPC protocols that are secure against active adversaries.

This section in organized as follows. Sections 5.1 and 5.2 contains (mostly standard) definitions of secure multiparty computation and secret sharing schemes, respectively. In Section 5.3 we prove that for n-party protocols that are secure in the case of an honest majority, it is enough to only consider adversaries controlling exactly t parties, where $n = 2t + 1$ (i.e., adversaries that control a maximal number of parties). In Section 5.4 we present the notions of *linear-based* protocols and weakly-private protocols. Next, in Section 5.5 we prove that any deviations of an active adversary from a weakly-private and linear-based protocol computing a circuit C in fact correspond to an additive attack on C. Finally, in Section 5.6 we generically construct MPC protocols that are secure in the presence of active adversaries from any linear-based and weakly-private protocol.

We then demonstrate our approach on many classical MPC protocols. In Section 5.7 we construct a protocol for securely computing an arithmetic circuit C in the pretense of an honest majority that has a communication complexity of $O(|C|n^2)$ field elements. We achieve this by proving that the semi-honest BGW protocol [BGW88, AL11] is in fact linear-based and weakly-private and then by applying our generic construction on it. Next, in Section 5.8, by proving that the semi-honest DN protocol [DN07] is also weakly-private and linear-based, we obtain a protocol for securely computing an arithmetic circuit

$|C|$ in the pretense of an honest majority that has a communication complexity of $O(|C|n + n^2)$ field elements.

Next, tackling the task of MPC without an honest majority, in Section 5.9 we demonstrate our methodology to an arithmetic version of the semi-honest GMW protocol [GMW87] and obtain a protocol for secure computation in the presence of active adversaries, which does not require an honest majority, in the OLE-hybrid model. Finally, in Section 5.10 we present a protocol for secure MPC without honest majority in the preprocessing model. Since the inputs to the preprocessing phase do not depend on the party's inputs to the protocol, it can be modeled as an additional party called the dealer that sends, to each party, the corresponding results of the preprocessing phase. Unlike many MPC protocols with preprocessing that require the dealer to behave honestly, we achieve a stronger security notion where either the dealer or any of the other parties may be corrupted (though not both).

5.1 Definitions

We start by defining a suitable notion of MPC. Our definitions are similar to [GIKR02] (for the protocol and adversary) and [Gol04, AL11] (for the security notion), with the addition of explicit adversarial inputs to the functionality. See [Can00, Gol04, AL11] for a more complete discussion. We also recall definitions of the MPC hybrid model and of the MPC client-server model [CDI05, DI05, DI06].

In order to simplify the protocol description and analysis, we treat functionalities, protocols, and adversaries as concrete finite objects and do not refer to infinite families of such objects. For this reason, we do not impose any efficiency requirement on simulators and do not refer to auxiliary inputs (unlike the standard definitions we cite). However, all of our protocols and simulators do in fact satisfy the additional requirements made in standard definitions.

We consider a network of n *parties* denoted by $P_1, \cdots, P_n$. Each pair of parties is connected via a private, authenticated point-to-point channel.

Functionality. A secure computation task is defined by an n-party functionality $f : X_1 \times \cdots \times X_n \to Y_1 \times \cdots Y_n$, specifying the desired mapping from parties inputs to their final outputs where for all i it holds that X_i and Y_i are the input and output domains of the i-th party. Sometimes we will allow an ideal-world adversary, called the simulator, to affect the functionality and change its internal logic. We achieve this by adding additional inputs to the functionality representing the simulator's attack on the functionality. In this case we define the functionality in the following way $f : X_1 \times \cdots \times X_n \times X_{\mathsf{atk}} \to Y_1 \times \cdots Y_n$ where as before for all i it holds that X_i and Y_i are the input and output domains of the i-th party and X_{atk} is the input domain of the simulator's attack on f. If the simulator is not allowed to attack the functionality we will omit these additional inputs from the description of the functionality.

In the sequel we focus on functionalities, with inputs and outputs that are vectors of field elements, where only the first party gets an output. Formally, we consider functionalities of the form $f : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_n} \times \mathbb{F}^{I_{\mathsf{atk}}} \to \mathbb{F}^{O_1}$ where $I_1, \cdots, I_n, I_{\mathsf{atk}}, O_1$ are positive integers. This is without loss of generality since any protocol π for securely computing single output functionalities can be used for computing functionaries where more than one party gets an output. We now briefly describe this transformation, see Chapter 7 in [Gol04] and Section 2.5.2 in [HL10] for more complete discussions. First, we transform a functionality f to a functionality f' where P_1 also gets the outputs of f for all the other parties but in a MACed and encrypted form, where the keys to the encryption and the MAC of each output are only known to the party that should get the output. Then, after invoking π in order to compute f', P_1 relays the corresponding outputs to all other parties that in turn decrypt and verify them. In case at least one of the MACs does not verify, all the parties abort the computation. Since the keys to the encryption and MAC are not known to P_1, he is unable to corrupt or learn the outputs of the other parties.

In the following we identify a circuit $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_m} \times \mathbb{F}^{I_{\mathsf{Adv}}} \to \mathbb{F}^{O_1}$ with the functionality it computes; the meaning will be clear from the context. An n-party circuit is a circuit computing an

n-party functionality.

We now define the notion of additively corruptible version of a circuit. Later, we will prove that any protocol for computing a functionality f that meets a list of certain requirement is a secure protocol for computing the additively corruptible version of f.

Definition 5.1 (Additively corruptible version of a circuit)**.** *Let $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_m} \to \mathbb{F}^{O_1}$ be an n-party circuit containing w wires. We define the* additively corruptible *version of C to be the n-party functionality $\widetilde{f}_C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_m} \times \mathbb{F}^w \to \mathbb{F}^{O_1}$ that takes additional input from the adversary which indicates an additive corruption for every wire of C. For all (x, A), $\widetilde{f}_C(x, A)$ outputs the result of the additively corrupted C as specified by the additive attack A (A is the simulator's attack on C) when invoked on the inputs x.*

Protocol. An n-party protocol is a collection of n parties where each party P_i holds an input x_i and random input r_i. The protocol proceeds in rounds where in each round i a protocol's description contains a description of n *next message functions* defined as follows. The next message function $\mathsf{nextMSG}_i^j$ of party P_i for the j-th round gets as input all the messages m_i^{j-1} that P_i received until the j-round, its input x_i to π, and its randomness r_i. Upon receiving its inputs, $\mathsf{nextMSG}_i^j$ outputs P_i's messages in the j-th round. For every party P_i the $\mathsf{nextMSG}_i$ function of the final round outputs P_i's output in π.

The *view* of the i-th party P_i during an execution of a protocol π on inputs $\vec{x}$, denoted by $\mathsf{view}_i^\pi(\vec{x})$, is defined as $(x_i, r_i, m_1, \cdots, m_\ell)$ where x_i is P_i's private input, r_i is its internal randomness and $m_1, \cdots, m_\ell$ are the messages it received during the protocol's execution. For every $T = \{i_1, \cdots, i_t\}$, we denote by $\mathsf{view}_T^\pi(\vec{x}) = (\mathsf{view}_{i_1}^\pi(\vec{x}), \cdots, \mathsf{view}_{i_t}^\pi(\vec{x}))$.

In the sequel we focus on protocols where all the next message functions can be represented as arithmetic circuits over some finite field $\mathbb{F}$.

Adversary. We consider an adversary Adv, corrupting up to t parties for some $t \in \mathbb{N}$. The adversary is a randomized interactive algorithm. It chooses a set T of at most t parties and corrupts them. The adversary then starts interacting with a protocol, where it takes control on over all parties in T. We will consider two types of adversaries. An *active* one, in addition to seeing the inputs and all messages sent to the parties it corrupted, is allowed to transmit any messages on behalf of these parties. A *passive* one is not allowed to change the messages transmitted by the corrupted parties; however, it still sees the inputs and all incoming messages transmitted to the corrupted parties. We assume by default that the adversary has a *rushing* capability: at any round it can first wait to hear all messages sent by the uncorrupted parties to parties in T and use these to determine its own messages. Finally, upon the protocol's termination, Adv outputs some function of its entire view.

Security. Informally, a protocol computing f is said to be t-secure if whatever an active adversary can "achieve" by corrupting at most t parties during the protocol execution, it could also achieve (by corrupting the same set of parties) in the ideal world in which f is evaluated using a trusted party. To formalize this definition, we need to define the concepts "achieve" and "ideal world". The *ideal world* process for evaluating the function f is a protocol involving all n parties and an additional, incorruptible, *trusted party*. The protocol is as follows. First, each party P_i and the adversary send their input x_i to the trusted party. Second, the trusted party computes f, and sends each party its corresponding output. Notice that when an adversary corrupts a party in the ideal process, it can pick the inputs sent by it to the trusted party.

For an adversary Adv we capture the notation of "achieves" in the real world by the random variable $\mathsf{Real}_{\pi,\mathsf{Adv},T}(\vec{x})$ consisting of the view of the adversary Adv controlling the corrupted parties in T and the outputs of the honest parties, following a real execution of π where for every $i \in \{1, \cdots, n\}$, party P_i has input x_i. Similarly, for an ideal world adversary Sim (the *simulator*) we capture the notion of "achieves" by the random variable $\mathsf{Ideal}_{f,\mathsf{Sim},T}(\vec{x})$ containing the outputs of the ideal adversary Sim and the honest parties after an ideal execution with the trusted party computing f where Sim has control over the

adversary's input of f. We are now ready to define what it means for a protocol to securely compute a functionality f in the presence of an adversary Adv. Intuitively, we would like it to be the case that anything achieved by an adversary Adv in the real world could be also achieved by the simulator in the ideal world. Formally,

Definition 5.2. *Let $f : X_1 \times \cdots \times X_n \times X_{\mathsf{atk}} \to Y_1 \times \cdots Y_n$ be a functionality and let π be an n-party protocol. We say that π (t, ϵ)-securely computes f if for every probabilistic adversary Adv in the real world controlling a set of parties $T \subseteq \{P_1, \cdots, P_n\}$ such that $|T| \leq t$, there exists a probabilistic simulator Sim in the ideal world such that for every input $\vec{x} \in X_1 \times \cdots \times X_n$ it holds that*

$$SD\left(\mathsf{Ideal}_{f,\mathsf{Sim},T}(\vec{x}), \mathsf{Real}_{\pi,\mathsf{Adv},T}(\vec{x})\right) \leq \epsilon.$$

If the protocol has the above property only for passive adversaries and simulators that do not deviate from the protocol we say that the protocol (t, ϵ)-privately computes f. In addition, for the case where $\epsilon = 0$ we say that the protocol t-securely (resp. t-privately) computes f.

Next, we define a relaxed notion of securely computing a functionality with abort. We will consider the weaker variant of security with abort (sometimes referred to as security with "selective abort") which does not require honest parties to simultaneously abort. This is captured by allowing the ideal world adversary to individually decide, after learning its own outputs, whether each honest party P_i receives its correct output from the functionality or a special "abort" message which P_i outputs. In the stronger notion of *unanimous abort*, the ideal world adversary needs to decide whether all honest parties receive their correct output or all of them abort.

The advantage of using the weaker variant is that it supports feasibility results which only use secure point-to-point channels and no additional setup. On the other hand, given a broadcast channel, any protocol Π realizing f with selective abort can be easily converted into one realizing f with "unanimous abort" by having each honest party broadcast a complaint message if Π leads it to abort. Such a complaint message makes all honest parties abort.

Security-with-abort. Let $f : X_1 \times \cdots \times X_n \times X_{\mathsf{atk}} \to Y_1 \times \cdots Y_n$ be a functionality. Consider a modified ideal world, called ideal world with abort, where the trusted party, after computing f, first sends the adversary its outputs and then the adversary decides, for each P_i, whether P_i should output its correct output y_i or a special abort symbol $\perp$. Let Sim be an ideal world with abort adversary. We denote the output of Sim and the honest parties after an ideal world execution with abort where the trusted party computes f on input $\vec{x}$ by $\mathsf{Ideal}^{abort}_{f,\mathsf{Sim},T}(\vec{x})$.

We now define what it means for a protocol π to be secure with abort.

Definition 5.3. *Let f be a functionality, if a protocol π has all the properties of Definition 5.2 where $\mathsf{Ideal}_{f,\mathsf{Sim},T}(\vec{x})$ is replaced with $\mathsf{Ideal}^{abort}_{f,\mathsf{Sim},T}(\vec{x})$. In this case we say that π (t, ϵ)-computes f with* abort.

The hybrid model. Following [Can00] we proceed by defining the (simplified) hybrid model for evaluating an n-party functionality f with the assistance of a trusted party evaluating an n-party functionality g. We will also define secure protocols in this model. The *hybrid model with ideal access to g*, denoted by g-hybrid model, is obtained as follows. We start from the real world process defined previously. We augment this process with an incorruptible trusted party for evaluating g. The trusted party is invoked at special rounds as determined by the protocol. The computation at each special round mimics the ideal process. That is, all parties send their inputs for g to the trusted party. As in the ideal process, an active adversary decides on the input values that the corrupted parties send to the trusted party. Since passive adversaries are not allowed to deviate from the protocol, in the case of passive adversaries, even corrupted parties send the trusted party their inputs according to the protocol. Next, the trusted party computes g using the inputs it was given. Finally, the trusted party send each party the corresponding output of g. For the case that g is a randomized-functionally, the

trusted party uses fresh randomness for each computation of g. Similarly to the standard model, for an adversary Adv we capture the notation of the "achieves" in the g-hybrid model by the random variable $\mathsf{Real}^{g}_{\pi,\mathsf{Adv},T}(\vec{x})$ consisting of the view of the adversary Adv controlling the corrupted parties in T and the outputs of the honest parties, following a real execution of π in the g-hybrid model where for every $i \in \{1,\cdots,n\}$, party P_i has an input x_i.

The following composition theorem is an extension of [Can00] to statically secure protocols.

Theorem 5.1. *Let t, n be positive integers such that $t < n$ and let f, g be n-party functionalities. Let π be an n-party protocol that (t,ϵ)-securely computes f with abort in the g-hybrid model where no more than one ideal evaluation call is made at each round and at most ℓ invocations of g are performed in total. In addition, let ρ be n-party protocols such that ρ (t,ϵ')-securely computes g with abort. Consider the protocol π^{ρ} (in the plain model) obtained by replacing each ideal call to g by the protocol ρ. Then it holds that the protocol π^{ρ} $(t,\epsilon+\ell\epsilon')$-securely computes f with abort in the plain model.*

The client-server model. We now describe a refinement of the standard MPC model called the client-server model (see [CDI05, DI05, DI06] for a detailed discussions). In the client-server model, each party can have one of two different roles: *clients* that hold inputs and get outputs, and *servers* who may be involved in the computation but hold no inputs and get no outputs. In the following, we denote by m the number of clients and by n the number of servers (rather than the total number of parties). If a protocol π is secure against any active adversary controlling at most s servers and at most t clients we say that π is (s,t,ϵ)-secure. Similarly, if π is private against any passive adversary controlling at most s servers and at most t clients we say that π is (s,t,ϵ)-private. In case the adversary is allowed to control an arbitrary number of clients, we will omit t from the notation.

Finally, notice that every protocol in the client-server model can be stated as a protocol in the standard MPC model by asking every party to play a single server and a single client. Below, the term m-client circuit relates to a circuit computing a functionality that gets its inputs form m different clients.

Notation. We now proceed to introduce some notation that we will be used throughout this section. Let π be a protocol computing some functionality f and let Adv be an adversary controlling a set of clients $\mathcal{C}$ and a set of servers $\mathcal{S}$. For a sets of parties T, T' (consisting of both clients and servers) we denote by $\widetilde{M}^{\pi,i,\mathsf{Adv}}_{T,T'}$ the random variables corresponding to messages sent from T to T' during the i-th round in a real execution of the protocol in the presence of Adv. Denote by $\widetilde{M}^{\pi,\mathsf{Adv}}_{T,T'}$ all the messages sent from T to T' during a real execution of π in the presence of Adv. Also, denote by $M^{\pi,\mathsf{honest}}_{T,T'}$ all the messages sent from T to T' during an honest execution of π.

In addition, denote by $V^{\pi,\mathsf{Adv}}_{T}$ the view of the parties in T during a real execution of the protocol in the presence of Adv, and denote by $O^{\pi,\mathsf{Adv}}_{T}$ the outputs of the parties in T after a real execution of π in the presence of Adv. By definition, it holds that $\mathsf{Real}_{\pi,\mathsf{Adv},\mathcal{S}\cup\mathcal{C}}(\vec{x}) \equiv (V^{\pi,\mathsf{Adv}}_{\mathcal{S}\cup\mathcal{C}}, O^{\pi,\mathsf{Adv}}_{\overline{\mathcal{C}}})$. Let $U^{\pi,\mathsf{Adv}}_{\mathcal{S}\cup\mathcal{C}}$ be the *truncated view* of the clients and servers in $\mathcal{S}\cup\mathcal{C}$ during a real execution of π in the presence of Adv excluding the last communication round. Let $L^{\pi,\mathsf{Adv}}_{\overline{\mathcal{S}},\mathcal{C}}$ be the messages sent to the clients in $\mathcal{C}$ during the last communication round during a real execution of π in the presence of Adv. Thus we have that $V^{\pi,\mathsf{Adv}}_{\mathcal{S}\cup\mathcal{C}} \equiv (U^{\pi,\mathsf{Adv}}_{\mathcal{S}\cup\mathcal{C}}, L^{\pi,\mathsf{Adv}}_{\overline{\mathcal{S}},\mathcal{C}})$.

Let Sim be an ideal world adversary corrupting a set of servers $\mathcal{S}$ and a set of clients $\mathcal{C}$. Analogously to the above, denote by $V'^{\pi,\mathsf{Sim}}_{\mathcal{S}\cup\mathcal{C}}$ the view simulated by Sim and denote by $O'^{\pi,\mathsf{Sim}}_{\overline{\mathcal{C}}}$ the outputs of the clients in $\overline{\mathcal{C}}$ during an ideal execution of the protocol. By definition it holds that $\mathsf{Ideal}_{f,\mathsf{Sim},\mathcal{S}\cup\mathcal{C}}(\vec{x}) \equiv (V'^{\pi,\mathsf{Sim}}_{\mathcal{S}\cup\mathcal{C}}, O'^{\pi,\mathsf{Sim}}_{\overline{\mathcal{C}}})$. Let $U'^{\pi,\mathsf{Sim}}_{\mathcal{S}\cup\mathcal{C}}$ be the *truncated view* simulated by Sim of the clients and servers in $\mathcal{S}\cup\mathcal{C}$ excluding the last communication round and let $L'^{\pi,\mathsf{Sim}}_{\overline{\mathcal{S}},\mathcal{C}}$ be the simulated messages by Sim to the clients in $\mathcal{C}$ during the last communication round. Thus we have that $V'^{\pi,\mathsf{Sim}}_{\mathcal{S}\cup\mathcal{C}} \equiv (U'^{\pi,\mathsf{Sim}}_{\mathcal{S}\cup\mathcal{C}}, L'^{\pi,\mathsf{Sim}}_{\overline{\mathcal{S}},\mathcal{C}})$.

In the squeal we will sometimes omit the Sim, Adv and π superscripts when the meaning is clear from the context.

Remark 5.1. In the following, we do not aim at optimizing the round complexity of our constructions. It can be easily verified that all of our protocols below can be implemented so that their round complexity is linear in the circuit depth.

5.2 Secret sharing schemes and randomness extraction

Secret sharing scheme. A t-out-of-n secret sharing scheme takes as input a secret s from some input domain and outputs n shares, with the property that it is possible to efficiently reconstruct s from every subset of $t+1$ shares, but every subset of at most t shares reveals nothing about the secret s. The value t is called the *threshold* of the scheme.

A secret sharing scheme consists of two algorithms: the first algorithm, called the *sharing algorithm*, takes as input the secret s and the parameters t and n, and outputs n shares. The second algorithm, called the *recovery algorithm* takes as input $t+1$ shares and outputs a value s. It is required that the reconstruction of shares generated from a value s produces the same value s. Formally, consider Definition 5.4 below.

Definition 5.4. *A pair of algorithms* $(\mathsf{share}, \mathsf{recover})$ *where* share *is randomized and* $\mathsf{recover}$ *is deterministic are said to be a secret sharing scheme if the following conditions hold for every* $n, t \in \mathbb{N}$.

- Reconstruction. *For any set* $T \subseteq \{1, \cdots, n\}$ *such that* $|T| > t$ *and for any* $x \in \mathbb{F}$ *it holds that*

$$\Pr[\mathsf{recover}(\mathsf{share}_T(x, n, t), T) = x] = 1$$

  where share_T *is the restriction of the outputs of* share *to the elements in* T.

- Privacy. *For any set* $T \subseteq \{1, \cdots, n\}$ *such that* $|T| \leq t$ *and for any* $x, y \in \mathbb{F}$ *it holds that*

$$\mathsf{share}_T(x, n, t) \equiv \mathsf{share}_T(y, n, t)$$

  where share_T *is the restriction of the outputs of* share *to the elements in* T.

In addition, we say that $(\mathsf{share}, \mathsf{recover})$ *is a* linear *secret sharing scheme if both* share *and* $\mathsf{recover}$ *compute their outputs using some fixed linear combination of their inputs and randomness.*

Also, we say that $(\mathsf{share}, \mathsf{recover})$ *is a* redundant *secret sharing scheme if for every pair of integers* n, t, *for every set* T *such that* $|T| > t$ *and for every input* x *it holds that for every* $\vec{v} \leftarrow \mathsf{share}(x, n, t)$ *the elements in* $\vec{v}_T$ *uniquely determine all the elements in* $\vec{v}$.

Finally, we say that $(\mathsf{share}, \mathsf{recover})$ *is a* dense *secret sharing scheme if for every pair of integers* n, t, *a vector* $\vec{v} \in \mathbb{F}^{t+1}$ *and for any set* T *such that* $|T| = t+1$ *there exists* $x \in \mathbb{F}$ *and suitable randomness of* share *such that* $\mathsf{share}_T(x, n, t)$ *outputs* $\vec{v}$.

The following corollary immediately follows from the privacy property of any secret sharing scheme.

Corollary 5.1. *Let* $(\mathsf{share}, \mathsf{recover})$ *be a secret sharing scheme and let* $n, t \in \mathbb{N}$. *Then, for any set of parties* T *such that* $|T| \leq t$ *and for any vector* $\vec{v}_T \in \mathbb{F}^t$ *such that* $\vec{v}_T \leftarrow \mathsf{share}_T(x, n, t)$ *for some* $x \in \mathbb{F}$ *and randomness for* share *it holds that for any* $y \in \mathbb{F}$ *there exists a vector* $\vec{v}_{\overline{T}} \in \mathbb{F}^{n-t}$ *such that* $y \leftarrow \mathsf{recover}((\vec{v}_T, \vec{v}_{\overline{T}}), T \cup \overline{T})$.

Shamir's secret-sharing scheme. Informally, Shamir's secret-sharing scheme works as follows (see [Sha79] and Section 3 in [AL11] for details). Let n, t be positive integers such that $t < n$, $\mathbb{F}$ be a finite field such that $|\mathbb{F}| > n$ and let $s \in \mathbb{F}$. The sharing algorithm generates a polynomial q

of degree at most t in $\mathbb{F}[x]$, such the free coefficient is set to be the secret s. The shares are defined to be $q(\alpha_i)$ for every $i \in \{1, \cdots, n\}$ where $\alpha_1, \cdots, \alpha_n$ are any distinct non-zero elements of $\mathbb{F}$. The reconstruction algorithm of the scheme is based on the fact that any $t+1$ points define exactly one polynomial of degree t. Thus, using Lagrange interpolation, it is possible to efficiently reconstruct the polynomial $q(x)$ given any subset of $t+1$ points as computed by the sharing algorithm. Then, given $q(x)$ is it possible to compute $s = q(0)$. We begin with the following definition of Lagrange interpolation points and coefficients.

Definition 5.5 (Lagrange coefficients)**.** *Let $\mathbb{F}$ be a finite field n, t positive integers such that $n = 2t+1$, $S \subseteq \{1, \cdots, n\}$ be a set of integers such that $|S| = t+1$ and let $(\alpha_1 \cdots, \alpha_n)$ be distinct non-zero field elements. We say that a vector $(\Delta_1^S, \cdots, \Delta_{t+1}^S)$ consists of the Lagrange coefficients corresponding to $(\alpha_1 \cdots, \alpha_n)$ and S if for every degree t polynomial q it holds that*

$$\sum_{i \in S} \Delta_i^S q(\alpha_i) = q(0).$$

Similarly, we say that a vector $(\Delta_1, \cdots, \Delta_n)$ consists of the degree $2t$ Lagrange coefficients corresponding to $(\alpha_1 \cdots, \alpha_n)$ if for every degree $2t$ polynomial q it holds that $\sum_{i=1}^n \Delta_i q(\alpha_i) = q(0)$.

We now present the notion of Shamir secret sharing scheme (see [Sha79] and Section 3 in [AL11] for details). Formally, consider Construction 5.1 below.

Construction 5.1 (Shamir secret sharing)**.** *Let n be a positive integer and let $\mathbb{F}$ be a finite field such that $|\mathbb{F}| > n$. In addition, let $\alpha_1, \cdots, \alpha_n$ be any distinct non zero elements of $\mathbb{F}$. Consider the algorithms* share *and* recover *defined below.*

- *The algorithm* share *on inputs $s, t, (\alpha_1, \cdots, \alpha_n)$ where $\alpha_1, \cdots, \alpha_n \in \mathbb{F}$ and $t < n$ performs the following:*
  - *Generate $q_1, \cdots, q_t$ uniformly at random from $\mathbb{F}$.*
  - *Define $q(x) = s + q_1 x + \cdots + q_t x^t$.*
  - *Output $(q(\alpha_1), \cdots, q(\alpha_n))$ where $q(\alpha_i)$ is the share of the ith party.*
- *The recovery algorithm* recover *gets as input a set S of $t+1$ points of the form (α_i, β_i). Then it reconstructs the* unique *polynomial $q(x)$ of degree t such that for all $1 \leq i \leq t+1$ it holds that $q(\alpha_i) = \beta_i$. Finally* recover *outputs $q(0)$.*

Randomness extraction. We define the notation of a *super invertible* matrix that will be useful later for the propose of randomness extraction.

Definition 5.6. *Let $\mathbb{F}$ be a finite field, $r > c$ be positive integers and let $M \in \mathbb{F}^{r \times c}$. We say that M is super invertible if for all $R \subseteq \{1, \cdots, r\}$ such that $|R| = c$ it holds that M_R is invertible.*

We now define the notation of R-random distributions.

Definition 5.7 (R-random distributions)**.** *Let r be a positive integers. For any $R \subseteq \{1, \cdots, r\}$ consider the distribution X_R of all vectors $(x_1, \cdots, x_r)$ over $\mathbb{F}^r$ that is defied as follows. For all $i \in R$ it holds that x_i is selected uniformly at random from $\mathbb{F}$ and for all $i' \in \{1, \cdots, r\} \setminus R$ it holds that $x_{i'}$ is generated with an arbitrary distribution that is independent from $\{x_i\}_{i \in R}$. In this case we say that X_R is R-random.*

We now claim that any super invertible matrix can be used in order to extract randomness from the distributions defined in Definition 5.7. Indeed, consider the following lemma.

Lemma 5.1. *Let $r > c$ be positive integers. We claim that for any super invertible matrix $M \in \mathbb{F}^{r\times c}$ and for any R-random distribution X_R over $\mathbb{F}^r$ such that $|R| \geq c$ it holds that*

$$M^T X_R \equiv U_c.$$

Proof. Let X_R be an R-random distribution over $\mathbb{F}^r$ such that $|R| \geq c$ and let $M \in \mathbb{F}^{r\times c}$ be a super invertible matrix. In addition, let Y be X_R restricted to the indices in R and let Y' be X_R restricted to the indices in $\{1, \cdots, r\} \setminus R$. Notice that Y and Y' are independent. Since X_R is an R-random distribution, we have that $Y \equiv U_r$. Thus, since M_R is invertible, we have that $(M_R)^T Y \equiv U_c$. Therefore, we obtain that,

$$M^T X_R \equiv (M_R)^T Y + (M_{\{1,\cdots,r\}\setminus R})^T Y' \equiv U_c + (M_{\{1,\cdots,r\}\setminus R})^T Y' \equiv U_c.$$

■

5.3 From general adversaries to maximal adversaries

In this section we prove that any MPC protocol in the client-server model that is secure against an adversary controlling exactly t servers is also secure against any adversary controlling at most t servers. This claim will be useful for us in Section 5.5 where we prove that private MPC protocols for computing a circuit C using $n = 2t + 1$ servers that have certain syntactical properties are secure protocols for computing the additively corruptible version $\widetilde{f_C}$ of C in the presence of an active adversary. Concretely, in each round of some of the protocols discussed in Section 5.5, the messages sent by each server to the other servers (including himself) are always Shamir sharings with threshold t of some value x. Thus, in case the adversary controllers exactly t servers, in each round, the simulator always gets, from every corrupted server, exactly $t+1$ shares from the adversary (one for each honest server) which will always form a valid Shamir sharing of some value x'. Thus, the simulator can recover x' and use it in order to simulate the output of the honest clients. This is in contrast to an adversary controlling less the t servers in which case the simulator gets at least $t + 2$ shares which might not form a valid shaming sharing (see Section 5.5 for a more detailed discussion).

Notice that in general, this is not true for the plain model. Let n, t be positive integers such that $t < n$ and let $f_{n,t}(x)$ be the functionality which on input x, obtained from P_1, computes an n-party Shamir secret sharing of x with threshold t and outputs each party, P_i, its corresponding share. Consider the protocol π for computing $f_{n,t}$ where P_1 in addition to sharing x between all the parties using Shamir secret sharing with threshold t also sends x itself to all the parties. Notice that π is a secure protocol for computing $f_{n,t}(x)$ against any adversary controlling exactly $t + 1$ parties. This is because any information obtained by the adversary during a real execution of π, can be simulated in the ideal world by recovering the $t + 1$ shares provided to the simulator by the trusted party. However, π is *not* secure against an adversary controlling t or less parties. This is since in the real world all the parties learn x which is not simulatable in the ideal world from t (or less) shares provided to the simulator by the trusted party.

We now claim that in the client server model, every protocol that is secure against any adversary controlling exactly t servers is also secure against any adversary controlling at most t servers.

Lemma 5.2. *Let Π be a protocol computing an m-client circuit C using $n = 2t + 1$ servers. Then, if Π is secure against any adversary controlling exactly t servers then Π is secure against any adversary controlling at most t servers.*

Proof. Let Π be a protocol computing an m-client circuit C using $n = 2t+1$ servers. Let $\{C_1, \cdots, C_m\}$ be the set of clients and let $\{S_1, \cdots, S_n\}$ be the set of servers. In addition let Adv be an adversary

controlling a set $\mathcal{C}$ of clients and a set $\mathcal{S}$ of servers such that $|\mathcal{S}| < t$. Finally, fix $\mathcal{S}^*$ to be some set of servers such that $|\mathcal{S}^* \cup \mathcal{S}| = t$.

Consider the adversary Adv^* controlling the clients in $\mathcal{C}$ and the servers in $\mathcal{S}^* \cup \mathcal{S}$ that initializes the servers in $\mathcal{S}^*$ as specified by Π and behaves as follows.

1. Adv^* simulates the honest behavior of the servers in $\mathcal{S}^*$. In particular, when the servers in $\mathcal{S}^*$ want to send a message m to the servers or clients in $\mathcal{S} \cup \mathcal{C}$, Adv^* sends m internally to Adv.

2. Adv^* simulates Adv on the servers and clients in $\mathcal{S} \cup \mathcal{C}$. In particular, when Adv wants to send a message m to the servers in $\mathcal{S}^*$, Adv^* sends m internally to $\mathcal{S}^*$.

Notice that Adv^* controls exactly t servers therefore Π is secure against Adv^*. Let Sim^* be a simulator as guaranteed by the security of Π against Adv^*.

Let $v_{\mathcal{S} \cup \mathcal{S}^* \cup \mathcal{C}}$ be a view of the parties in $\mathcal{S} \cup \mathcal{S}^* \cup \mathcal{C}$ and let $\mathsf{fixView}(v_{\mathcal{S} \cup \mathcal{S}^* \cup \mathcal{C}}, \mathcal{S}^*, m_{\mathcal{S}^*, \mathcal{S} \cup \mathcal{C}})$ be a function that removes from $v_{\mathcal{S} \cup \mathcal{S}^* \cup \mathcal{C}}$ the incoming messages to the servers in $\mathcal{S}^*$ and adds to $v_{\mathcal{S} \cup \mathcal{S}^* \cup \mathcal{C}}$ the messages $m_{\mathcal{S}^*, \mathcal{S} \cup \mathcal{C}}$ sent by the servers in $\mathcal{S}^*$ to the clients and servers in $\mathcal{S} \cup \mathcal{C}$. We use Sim^* and $\mathsf{fixView}$ to construct a simulator Sim for Adv that, on inputs $x_{\mathcal{C}}$ for the clients in $\mathcal{C}$, performs the following.

1. Invoke Sim^* on the inputs $x_{\mathcal{C}}$ and obtain a view v'^* for Adv^*. In addition, Sim calls the trusted party computing C whenever Sim^* calls the trusted party.

2. Invoke Adv^* on the view v'^* and obtain the messages $m'_{\mathcal{S}^*, \mathcal{S} \cup \mathcal{C}}$ sent by the servers in $\mathcal{S}^*$ to the clients and servers in $\mathcal{S} \cup \mathcal{C}$. Note that $m'_{\mathcal{S}^*, \mathcal{S} \cup \mathcal{C}}$ are well defined since Adv^* is deterministic given the internal randomness in v'^*.

3. Remove from v'^* the incoming messages to the servers in $\mathcal{S}^*$ as well the randomness used by the servers in $\mathcal{S}^*$. In addition, add $m'_{\mathcal{S}^*, \mathcal{S} \cup \mathcal{C}}$ to v'^*. That is, compute $v'_{\mathcal{S} \cup \mathcal{C}} \leftarrow \mathsf{fixView}(v'^*, \mathcal{S}^*, m'_{\mathcal{S}^*, \mathcal{S} \cup \mathcal{C}})$.

4. Output $v'_{\mathcal{S} \cup \mathcal{C}}$.

We now claim that Sim correctly simulates the view of Adv. Formally we claim that for any $\vec{x}$ it holds that

$$\mathsf{Ideal}_{\widetilde{f}_C, \mathsf{Sim}, \mathcal{S} \cup \mathcal{C}}(\vec{x}) \equiv \mathsf{Real}_{\Pi, \mathsf{Adv}, \mathcal{S} \cup \mathcal{C}}(\vec{x}).$$

Notice that by the security properties of Sim^* we have that for any $\vec{x}$ it holds that

$$\mathsf{Ideal}_{\widetilde{f}_C, \mathsf{Sim}^*, \mathcal{S} \cup \mathcal{S}' \cup \mathcal{C}}(\vec{x}) \equiv \mathsf{Real}_{\Pi, \mathsf{Adv}^*, \mathcal{S} \cup \mathcal{S}' \cup \mathcal{C}}(\vec{x}). \tag{23}$$

We begin by claiming that for any input $\vec{x}$ it holds that $V'^{\mathsf{Sim}}_{\mathcal{S} \cup \mathcal{C}} \equiv V^{\mathsf{Adv}}_{\mathcal{S} \cup \mathcal{C}}$.

Indeed, assume Adv is deterministic and let $\vec{x}$ be an input to C. Notice that by Equation 23 it holds that $V^{\mathsf{Adv}^*}_{\mathcal{S} \cup \mathcal{S}^* \cup \mathcal{C}} \equiv V'^{\mathsf{Sim}^*}_{\mathcal{S} \cup \mathcal{S}^* \cup \mathcal{C}}$. Denote by $M'^{\mathsf{Sim}^*}_{\mathcal{S}^*, \mathcal{S} \cup \mathcal{C}}$ the messages obtained in Step 2 of Sim when invoked on inputs $x_{\mathcal{C}}$ for the clients in $\mathcal{C}$. Finally, let $\mathsf{Adv}^*{}_{\mathcal{S} \cup \mathcal{C}}(v_{\mathcal{S} \cup \mathcal{S}^* \cup \mathcal{C}})$ be the function computing the messages sent to the servers and clients in $\mathcal{S} \cup \mathcal{C}$ during Step 1 inside Adv^* given its view $v_{\mathcal{S} \cup \mathcal{S}^* \cup \mathcal{C}}$. Notice that since Adv^* is completely deterministic given its view $v_{\mathcal{S} \cup \mathcal{S}^* \cup \mathcal{C}}$ it holds that,

$$\widetilde{M}^{\mathsf{Adv}^*}_{\mathcal{S}^*, \mathcal{S} \cup \mathcal{C}} \equiv \mathsf{Adv}^*{}_{\mathcal{S} \cup \mathcal{C}}(V^{\mathsf{Adv}^*}_{\mathcal{S} \cup \mathcal{S}^* \cup \mathcal{C}}) \text{ and } M'^{\mathsf{Sim}^*}_{\mathcal{S}^*, \mathcal{S} \cup \mathcal{C}} \equiv \mathsf{Adv}^*{}_{\mathcal{S} \cup \mathcal{C}}(V'^{\mathsf{Sim}^*}_{\mathcal{S} \cup \mathcal{S}^* \cup \mathcal{C}}).$$

Thus, since $V^{\mathsf{Adv}^*}_{\mathcal{S} \cup \mathcal{S}^* \cup \mathcal{C}} \equiv V'^{\mathsf{Sim}^*}_{\mathcal{S} \cup \mathcal{S}^* \cup \mathcal{C}}$ it holds that $(V'^{\mathsf{Sim}^*}_{\mathcal{S} \cup \mathcal{S}^* \cup \mathcal{C}}, M'^{\mathsf{Sim}^*}_{\mathcal{S}^*, \mathcal{S} \cup \mathcal{C}}) \equiv (V^{\mathsf{Adv}^*}_{\mathcal{S} \cup \mathcal{S}^* \cup \mathcal{C}}, \widetilde{M}^{\mathsf{Adv}^*}_{\mathcal{S}^*, \mathcal{S} \cup \mathcal{C}})$. Finally, notice that in a real execution of Π in the presence of Adv, it holds that $V^{\mathsf{Adv}}_{\mathcal{S} \cup \mathcal{C}} \equiv \mathsf{fixView}(V^{\mathsf{Adv}^*}_{\mathcal{S} \cup \mathcal{S}^* \cup \mathcal{C}}, \mathcal{S}^*, \widetilde{M}^{\mathsf{Adv}^*}_{\mathcal{S}^*, \mathcal{S} \cup \mathcal{C}})$. Similarly, by construction, $V'^{\mathsf{Sim}}_{\mathcal{S} \cup \mathcal{C}} \equiv \mathsf{fixView}(V'^{\mathsf{Sim}^*}_{\mathcal{S} \cup \mathcal{S}^* \cup \mathcal{C}}, \mathcal{S}^*, M'^{\mathsf{Sim}^*}_{\mathcal{S}^*, \mathcal{S} \cup \mathcal{C}})$.

Thus we have that,

$$\begin{aligned} V'^{\mathsf{Sim}}_{\mathcal{S} \cup \mathcal{C}} &\equiv \mathsf{fixView}(V'^{\mathsf{Sim}^*}_{\mathcal{S} \cup \mathcal{S}^* \cup \mathcal{C}}, \mathcal{S}^*, M'^{\mathsf{Sim}^*}_{\mathcal{S}^*, \mathcal{S} \cup \mathcal{C}}) \\ &\equiv \mathsf{fixView}(V^{\mathsf{Adv}^*}_{\mathcal{S} \cup \mathcal{S}^* \cup \mathcal{C}}, \mathcal{S}^*, \widetilde{M}^{\mathsf{Adv}^*}_{\mathcal{S}^*, \mathcal{S} \cup \mathcal{C}}) \\ &\equiv V^{\mathsf{Adv}}_{\mathcal{S} \cup \mathcal{C}}. \end{aligned}$$

We now set out to prove that $(V'^{\mathsf{Sim}}_{\mathcal{S}\cup\mathcal{C}}, O'^{\mathsf{Sim}}_{\overline{\mathcal{C}}}) \equiv (V^{\mathsf{Adv}}_{\mathcal{S}\cup\mathcal{C}}, O^{\mathsf{Adv}}_{\overline{\mathcal{C}}})$. Denote by $\Pi_{\overline{\mathcal{C}}}(x_{\overline{\mathcal{C}}}, m_{\overline{\mathcal{C}}})$ the outputs of the clients in $\overline{C}$ during a real execution of Π given their inputs $x_{\overline{\mathcal{C}}}$ and the messages they received $m_{\overline{\mathcal{C}}}$. By definition of $\Pi_{\overline{\mathcal{C}}}(x_{\overline{\mathcal{C}}}, m_{\overline{\mathcal{C}}})$ we have that,

$$(V^{\mathsf{Adv}}_{\mathcal{S}\cup\mathcal{C}}, O^{\mathsf{Adv}}_{\overline{\mathcal{C}}}) \equiv (V^{\mathsf{Adv}}_{\mathcal{S}\cup\mathcal{C}}, \Pi_{\overline{\mathcal{C}}}(x_{\overline{\mathcal{C}}}, \widetilde{M}^{\mathsf{Adv}}_{\overline{\mathcal{C}}})).$$

Notice that by construction of Adv^*, it holds that the messages sent by Adv^* and Adv to the honest clients and servers are the same. Thus using the definition of $\mathsf{fixView}$ we have that,

$$(V^{\mathsf{Adv}}_{\mathcal{S}\cup\mathcal{C}}, \Pi_{\overline{\mathcal{C}}}(x_{\overline{\mathcal{C}}}, \widetilde{M}^{\mathsf{Adv}}_{\overline{\mathcal{C}}})) \equiv (\mathsf{fixView}(V^{\mathsf{Adv}^*}_{\mathcal{S}\cup\mathcal{S}^*\cup\mathcal{C}}, \mathcal{S}^*, \widetilde{M}^{\mathsf{Adv}^*}_{\mathcal{S}^*,\mathcal{S}\cup\mathcal{C}}), \Pi_{\overline{\mathcal{C}}}(x_{\overline{\mathcal{C}}}, \widetilde{M}^{\mathsf{Adv}^*}_{\overline{\mathcal{C}}})).$$

Using the definition of $O^{\mathsf{Adv}^*}_{\overline{\mathcal{C}}}$ and the security properties of Sim^* we have that,

$$\begin{aligned}(\mathsf{fixView}(V^{\mathsf{Adv}^*}_{\mathcal{S}\cup\mathcal{S}^*\cup\mathcal{C}}, \mathcal{S}^*, \widetilde{M}^{\mathsf{Adv}^*}_{\mathcal{S}^*,\mathcal{S}\cup\mathcal{C}}), \Pi_{\overline{\mathcal{C}}}(x_{\overline{\mathcal{C}}}, \widetilde{M}^{\mathsf{Adv}^*}_{\overline{\mathcal{C}}})) &\equiv (\mathsf{fixView}(V^{\mathsf{Adv}^*}_{\mathcal{S}\cup\mathcal{S}^*\cup\mathcal{C}}, \mathcal{S}^*, \widetilde{M}^{\mathsf{Adv}^*}_{\mathcal{S}^*,\mathcal{S}\cup\mathcal{C}}), O^{\mathsf{Adv}^*}_{\overline{\mathcal{C}}}) \\ &\equiv (\mathsf{fixView}(V'^{\mathsf{Sim}^*}_{\mathcal{S}\cup\mathcal{S}^*\cup\mathcal{C}}, \mathcal{S}^*, M'^{\mathsf{Sim}^*}_{\mathcal{S}^*,\mathcal{S}\cup\mathcal{C}}), O'^{\mathsf{Sim}^*}_{\overline{\mathcal{C}}})\end{aligned}$$

Next, notice that Sim and Sim^* make the same calls to the trusted party. Thus using the definition of $\mathsf{fixView}$ we have that,

$$(\mathsf{fixView}(V'^{\mathsf{Sim}^*}_{\mathcal{S}\cup\mathcal{S}^*\cup\mathcal{C}}, \mathcal{S}^*, M'^{\mathsf{Sim}^*}_{\mathcal{S}^*,\mathcal{S}\cup\mathcal{C}}), O'^{\mathsf{Sim}^*}_{\overline{\mathcal{C}}}) \equiv (V'^{\mathsf{Sim}}_{\mathcal{S}\cup\mathcal{C}}, O'^{\mathsf{Sim}}_{\overline{\mathcal{C}}}).$$

Thus we have obtained, $(V^{\mathsf{Adv}}_{\mathcal{S}\cup\mathcal{C}}, O^{\mathsf{Adv}}_{\overline{\mathcal{C}}}) \equiv (V'^{\mathsf{Sim}}_{\mathcal{S}\cup\mathcal{C}}, O'^{\mathsf{Sim}}_{\overline{\mathcal{C}}})$ meaning that Π is a secure protocol for computing C in the precise of Adv. ■

5.4 Linear-based protocols

In this section we describe a set of properties we will require from an MPC protocol Π computing a circuit C. Later, we will prove that if Π meets these properties, then Π is a secure protocol for computing the additively corruptible version $\widetilde{f}_C$ of C as defined in Definition 5.1. We begin by defining the notion of *linear protocol* below.

Definition 5.8 (Linear protocol)**.** *An n-party protocol Π is said to be* linear protocol*, over some finite field $\mathbb{F}$ if Π has the following properties*

1. **Inputs.** *The input of every party P_i is a vector of field elements from $\mathbb{F}$. Moreover, P_i's inputs can be divided into two distinct types, the main inputs and auxiliary inputs.*
2. **Messages.** *Recall that each message in Π is a vector of field elements. We require that every message $\vec{m}$ of Π, sent by some party P_i, belongs to one of the following categories:*
   (a) *$\vec{m}$ is some fixed arbitrary function of P_i's main inputs (and is independent of its auxiliary inputs).*
   (b) *every entry m_j of $\vec{m}$ is generated as some fixed linear combination of P_i's auxiliary inputs and elements of previous messages received by P_i.*
3. **Output.** *The output of every party P_i is a linear function of its incoming messages.*

Notice that for any linear protocol Π, input x and for any set of parties T, it is possible to compute the outputs of the parties in T given the messages $m_{\mathsf{inp},T}$ of type 2a in Definition 5.8 sent by the parties in T to themselves, the auxiliary input $\vec{y}$ as well as the incoming $m_{\overline{T},T}$ messages of type 2b sent by the parties in $\overline{T}$ to the parties in T. We thus define the notion of *output function of T in π* to be as follows.

Definition 5.9 (Output function of a linear protocol)**.** *Let π be a linear protocol for computing a functionality f and let T be a set of parties. Let $\vec{x}$ be a main input to π, let $\vec{y}$ be an auxiliary input to π and let $m_{\mathsf{inp},T}$ be the messages of type 2a in Definition 5.8 sent by the parties in T to themselves during an honest execution of π on $(\vec{x},\vec{y})$. In addition, let $m_{\overline{T},T}$ be the messages of type 2b sent by the parties in $\overline{T}$ to the parties in T during an honest execution of π. We say that a function output_T is the* output function of T in π *if for any main input $\vec{x}$ and auxiliary input $\vec{y}$ it holds that*

$$\mathsf{output}_T(m_{\mathsf{inp},T},\vec{y},m_{\overline{T},T}) = f_T(\vec{x},\vec{y})$$

where f_T is the restriction of f to the outputs of the parties in T.

The following immediate claim states that the output function of a linear protocol is a linear function.

Claim 5.1. *Let π be a linear protocol and let T be a set of parties. In addition let output_T be the output function of T in π. Then for any m_1,y,m_2,m_1',y',m_3 it holds that*

$$\mathsf{output}_{T,\pi}(m_1+m_1',y+y',m_2+m_2') = \mathsf{output}_{T,\pi}(m_1,y,m_2)+\mathsf{output}_{T,\pi}(m_1',y',m_2').$$

Next, we define the notion of *linear-based* protocols which use linear protocols as internal components. Notice that, linear-based protocols in fact capture the typical structure of many common MPC protocols such as the BGW protocol [BGW88, AL11] and the DN protocol [DN07]. Later we will claim that any protocol π for computing an m-client circuit C using $n=2t+1$ servers that is linear-based and weakly-private against active adversaries controlling at most t servers is a t-secure protocol for computing the additively corruptible version $\widetilde{f_C}$ of C as defined in Definition 5.1. In particular, this observation will hold for the BGW and DN protocols.

We proceed by defining the notion of linear-based protocols below.

Definition 5.10 (Linear-based protocols)**.** *Let $\mathcal{SS}=(\mathsf{share},\mathsf{recover})$ be a redundant dense linear secret sharing scheme. An n-server m-client protocol Π for computing a single-output m-client circuit $C:\mathbb{F}^{I_1}\times\cdots\times\mathbb{F}^{I_m}\to\mathbb{F}^{O_1}$ where $n=2t+1$ is said to be* linear-based *with respect to $\mathcal{SS}$ if Π has the following structure:*

1. **Setup phase.** *During this phase all servers (and not the clients) participate in some linear protocol π_{setup} that gets no auxiliary inputs. At the end of this phase every server S_i holds a vector $\vec{h}_i^c$ for every multiplication gate g^c of C.*

2. **Randomness generation phase.** *During this phase all servers (and not the clients) participate in some linear protocol π_{rand} that gets no auxiliary inputs. At the end of this phase every server S_i holds a share g_i^j for every randomness gate g^j in C.*

3. **Input sharing phase.** Π *evaluates every input gate g^i belonging to client C_j of C as follows. C_j shares its input x for g^i using $\mathcal{SS}$ and then sends each server its corresponding share.*

4. **Circuit evaluation phase.** Π *computes C in stages. During the i-th stage in an honest execution, the i-th gate, g^i, inside C is evaluated (in some topological order) and at the end of the stage the servers hold a sharing of the output of g^i with a distribution induced by* share*. The evaluation of each gate is done as follows:*

   (a) *For any addition gate g^c in C with inputs g^a and g^b, Π evaluates g^c by having each server S_i sum its shares corresponding to the outputs of g^a and g^b. Similarly, for a subtraction gate, S_i subtracts the shares corresponding to the outputs of g^a and g^b. There is no communication during these stages.*

(b) *For any multiplication gate g^c in C with inputs g^a and g^b,* Π *evaluates g^c using some n-party linear protocol π_{mult} such that the main inputs of the i-th server S_i to π_{mult} are its shares g_i^a and g_i^b corresponding to the outputs of g^a and g^b. The auxiliary input of S_i to π_{mult} is $\vec{h}_i^c$ which is the results of the setup phase associated with g^c.*

5. **Output recovery phase.** *The output recovery phase of* Π *is done as follows: for each output gate of C, the first $t+1$ servers send their corresponding shares to C_1 which in turn recovers each output of C using* recover*.*

We now proceed to define the notion of *weakly-private* protocols. At high level, the notion of weak privacy is defined just like regular privacy in the sense that the view of any adversary can be simulated. However, the notion of *weakly-private* is weaker than regular privacy in the sense that it *does not* require the simulator to simulate the adversary's view during the last communication round of the protocol. We refer to the adversary's view excluding the last communication round as its *truncated view.*

Definition 5.11 (Weak privacy)**.** *Let π be an m-client protocol for computing a functionality f where only the first client gets an output and let* Adv *be an adversary controlling a set of clients $\mathcal{C}$. Denote by $\mathsf{view}_{\mathsf{Adv}}^{\pi,\mathsf{trunc}}(\vec{x})$ the view of* Adv *excluding the last communication round during a real execution of π on inputs $\vec{x}$. We say that π is* weakly-private *against* Adv *if there exists a simulator* Sim *such that for every input $\vec{x}$ it holds that*

$$\mathsf{view}_{\mathsf{Adv}}^{\pi,\mathsf{trunc}}(\vec{x}) \equiv \mathsf{Sim}(\vec{x}_{\mathcal{C}}).$$

Notice that Sim *does not have oracle access to a trusted party.*

The following theorem states that any protocol π for computing a circuit C that is linear-based with respect to some redundant and dense linear secret sharing scheme and that is weakly-private against active adversaries is a secure protocol for computing the additively corruptible version of C.

Theorem 5.2. *Let* Π *be a protocol computing a (possibly randomized) m-client circuit $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_m} \to \mathbb{F}^{O_1}$ using $n = 2t+1$ servers that is linear-based with respect to some redundant and dense linear secret sharing scheme and is weakly-private against active adversaries controlling at most t servers and an arbitrary number of clients. Then,* Π *is a t-secure protocol for computing $\widetilde{f}_C$.*

We prove Theorem 5.2 in two stages. First, using the results from Section 5.3, its enough to only consider adversaries that control exactly t servers and an arbitrary number of clients. Second, in Section 5.5 we prove that Π securely computes $\widetilde{f}_C$ in the presence of any active adversary controlling exactly t servers and an arbitrary number of clients.

5.5 Security of linear-based protocols

In this section we prove that any protocol Π for computing an m-client circuit C using $n = 2t+1$ servers that is linear-based and weakly-private against active adversaries controlling exactly t servers and an arbitrary number of clients is a secure protocol for computing the additively corruptible version $\widetilde{f}_C$ of C against active adversaries controlling exactly t servers.

Combining the results of this section with Lemma 5.2 we obtain that any protocol for computing an m-client circuit C using $n = 2t+1$ servers that is linear-based and weakly-private against active adversaries controlling at most t servers and an arbitrary number of clients is a t-secure protocol for computing the additively corruptible version $\widetilde{f}_C$ of C.

In order to prove that Π is a secure protocol from computing $\widetilde{f}_C$ against adversaries controlling exactly t servers we need to describe a simulator Sim for any such Adv. Recall that Sim must simulate the view of Adv together with the output of the honest parties. Notice that by the weak-privacy requirement it is possible to simulate the truncated view (the view excluding the last communication

round) of any active adversary controlling t servers. Simulating the output of the honest clients and the view of Adv during the last communication round is more challenging in the case of active adversaries.

In order to achieve this, Sim will use Adv in order to determine an appropriate additive attack on C. Recall that an additive assigns an element of $\mathbb{F}$ to each internal wire of C as well as to each of C's outputs. For an internal wire (a,b) in C, $A_{a,b}$ denotes the restriction of A to the wire (a,b). Similarly A_{out} denotes the restriction of A to the output wires of C. Using A, and using the output of the trusted party, the simulator will simulate the view of the adversary during the last communication round together with the output of the honest clients.

Notice that during the input sharing phase, the simulator has the messages that the adversary *actually* sent to the honest servers. In addition, since the simulator has the input of the corrupted clients, the simulator can compute by itself the messages that the adversary *should have* sent if it where behaving honestly. Next, as the simulation progress, for each multiplication gate, the simulator obtains the messages corresponding to the gate's output from the corrupted servers. In addition, the simulator will always will try to compute the so called "honest messages" that the adversary *should have* sent (based on it internal state), if it where to start behaving honestly at the current round. Finally, since Π evaluates multiplication gates using a linear protocol, the difference between the actual messages and the honest messages is used to determine the corresponding additive attack on the output of the multiplication gate. We will show by induction that thus computed additive attack A indeed "explains" all of the corruptions in the output of C_1 in case its honest and view of Adv in final round in the case where C_1 is corrupted.

We now proceed to formally describe a simulator for Π in the malicious model.

Construction 5.2. *Let Π be a linear-based protocol for computing a (possibly randomized) m-client circuit $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_m} \to \mathbb{F}^{O_1}$ using $n = 2t+1$ servers that is weakly-private against active adversaries controlling at most t servers and linear-based with respect to some redundant and dense linear secret sharing scheme $\mathcal{SS}$. In addition, let Adv be an adversary controlling a set $\mathcal{C}$ of clients and a set $\mathcal{S}$ of servers such that $|\mathcal{S}| = t$. Define the simulator Sim that on inputs $\vec{x}_{\mathcal{C}}$ (of the corrupted clients in $\mathcal{C}$), initializes an additive attack A and performs the following.*

1. **Truncated view generation phase.** *Let $\mathsf{Sim}_{\mathsf{trunc-view}}$ be a simulator as guaranteed by the weak-privacy property of Π. Invoke $\mathsf{Sim}_{\mathsf{trunc-view}}$ on the inputs $x_{\mathcal{C}}$ and obtain a simulated truncated view $u'_{\mathcal{S}\cup\mathcal{C}}$ of the adversary.*
2. **Setup phase.**[10] *Recall that by Definition 5.10 the protocol π_{setup} gets no auxiliary inputs. Let $\mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{setup}}}$ be the output function of $\overline{\mathcal{S}}$ in π_{setup} as defined in Definition 5.9. The simulation proceeds as follows.*
   (a) *Simulate the honest behavior of the servers in $\mathcal{S}$ given their truncated view $u'_{\mathcal{S}\cup\mathcal{C}}$ and obtain the messages $m'^{\pi_{\mathsf{setup}}}_{\mathcal{S},\overline{\mathcal{S}}}$ that should have been sent by the servers in $\mathcal{S}$ to the servers in $\overline{\mathcal{S}}$ during the execution of π_{setup}. In addition, for every server $S_i \in \mathcal{S}$ for every multiplication gate g^c obtain the vector $\vec{h}'^{c}_i$ that is a part of the output of S_i at the after an honest execution of π_{setup}.*
   (b) *Invoke Adv on the truncated view $u'_{\mathcal{S}\cup\mathcal{C}}$ and obtain the messages $\widetilde{m}'^{\pi_{\mathsf{setup}}}_{\mathcal{S},\overline{\mathcal{S}}}$ sent by the adversary to the servers in $\overline{\mathcal{S}}$ during the execution of π_{setup}.*
   (c) *Compute $\gamma^{\pi_{\mathsf{setup}}}_{\overline{\mathcal{S}}} \leftarrow \mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{setup}}}(0,\perp,\widetilde{m}'^{\pi_{\mathsf{setup}}}_{\mathcal{S},\overline{\mathcal{S}}} - m'^{\pi_{\mathsf{setup}}}_{\mathcal{S},\overline{\mathcal{S}}})$ where $\mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{setup}}}$ is the output function of π_{setup}.*

[10] For the case where Π does not have a setup phase, the results obtained from simulating this phase (namely $\gamma^{\pi_{\mathsf{setup}}}_{\overline{\mathcal{S}}}$ and $\vec{h}'_{\overline{\mathcal{S}}}$) should be considered as the empty vectors.

3. **Randomness generation phase.** *Recall that by Definition 5.10 the protocol π_{rand} gets no auxiliary inputs. Let $\mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{rand}}}$ be the output function of $\overline{\mathcal{S}}$ in π_{rand} as defined in Definition 5.9. The simulation proceeds as follows.*

   (a) *Simulate the honest behavior of the servers in $\mathcal{S}$ given their truncated view $u'_{\mathcal{S}\cup\mathcal{C}}$ and obtain the messages $m'^{\pi_{\mathsf{rand}}}_{\mathcal{S},\overline{\mathcal{S}}}$ that should have been sent by the servers in $\mathcal{S}$ to the servers in $\overline{\mathcal{S}}$ during the execution of π_{rand}. In addition, for every server $S_i \in \mathcal{S}$ for every randomness gate g^j obtain the share $\overline{g}'^j_i$ that is part of the output of S_i at the after an honest execution of π_{rand}.*

   (b) *Invoke Adv on the truncated view $u'_{\mathcal{S}\cup\mathcal{C}}$ and obtain the messages $\widetilde{m}'^{\pi_{\mathsf{rand}}}_{\mathcal{S},\overline{\mathcal{S}}}$ sent by the adversary to the servers in $\overline{\mathcal{S}}$ during the execution of π_{rand}.*

   (c) *Compute $\gamma^{\pi_{\mathsf{rand}}}_{\overline{\mathcal{S}}} \leftarrow \mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{rand}}}(0,\perp,\widetilde{m}'^{\pi_{\mathsf{rand}}}_{\mathcal{S},\overline{\mathcal{S}}} - m'^{\pi_{\mathsf{rand}}}_{\mathcal{S},\overline{\mathcal{S}}})$.*

   (d) *For every randomness gate g^c, let $\gamma^c_{\overline{\mathcal{S}}} \in \mathbb{F}^{t+1}$ be the restriction of $\gamma^{\pi_{\mathsf{rand}}}_{\overline{\mathcal{S}}}$ to the values corresponding to g^c. The simulation proceeds as follows.*

      i. *The simulator now determines entries of the additive attack A on the circuit C. Notice that $\gamma^c_{\overline{\mathcal{S}}}$ is a vector of $t+1$ shares of $\mathcal{SS}$ and thus, since $\mathcal{SS}$ is dense, this forms a valid sharing of some value.*
      Compute $\alpha^c \leftarrow \mathsf{recover}(\gamma^c_{\overline{\mathcal{S}}},\overline{\mathcal{S}})$ and every gate g^d connected to g^c set $A_{c,d} \leftarrow \alpha^c$. Finally, since $\mathcal{SS}$ is redundant, the simulator is able to compute the shares $(\gamma^c_i)_{S_i\in\mathcal{S}}$ for the servers in $\mathcal{S}$ that are compatible with $\gamma^c_{\overline{\mathcal{S}}}$.

      ii. *Next, the simulator proceeds to compute the shares $(g'^c_i)_{S_i\in\mathcal{S}}$. For every $S_i \in \mathcal{S}$ compute $g'^c_i \leftarrow \overline{g}'^c_i + \gamma^c_i$ and saves it for later use.*

4. **Input sharing phase.**

   (a) *For each input gate g^c that is part of the inputs of some honest client C_i:*

      i. *For every corrupted server S_k retrieve from $u'_{\mathcal{S}\cup\mathcal{C}}$ the value g'^c_k representing S_k's share of C_i'th input for g^c and send it to Adv.*

      ii. *For any gate g^d connected to the output of g^c set $A_{c,d} \leftarrow 0$.*

   (b) *For each input gate g^c belonging to the corrupted client $C_i \in \mathcal{C}$, the simulator performs the following:*

      i. *For each honest server $S_k \in \overline{\mathcal{S}}$, receive a message $\widetilde{g}'^c_k$ from Adv corresponding to S_k'th share of Adv's input for g^c.*

      ii. *Notice that since $|\overline{\mathcal{S}}| = t+1$ and since $\mathcal{SS}$ is a dense scheme, the messages obtained in Step 4(b)i above always form a secret sharing of some value $\widetilde{x}_c$. That is $\widetilde{x}_c \leftarrow \mathsf{recover}\left((\widetilde{g}'^c_k)_{S_k\in\overline{\mathcal{S}}},\overline{\mathcal{S}}\right)$. For every gate g^d connected to the output of g^c the simulator sets $A_{c,d} \leftarrow \widetilde{x}_c - x_c$ where x_c is the input of C_i to g^c.*

      iii. *Next, for each corrupted server S_k, using the redundancy property of $\mathcal{SS}$, the simulator computes the shares g'^c_k that is consistent with the shares obtained in Step 4(b)i above and stores it for later use.*

5. **Circuit evaluation phase.** *For each gate g^c of C with inputs g^a and g^b, proceed as follows:*

   Handling addition and subtraction gates. *In this case no communication takes place. For each corrupted server S_i compute $g'^c_i \leftarrow g'^a_i + g'^b_i$ in the case of addition gates and $g'^c_i \leftarrow g'^a_i - g'^b_i$ in the case of subtraction gates. Also, for every gate g^d connected to the output of g^c, set $A_{c,d} \leftarrow 0$.*

   Handling multiplication gates. *If g^c is a multiplication gate, recall that Π evaluates g^c using some linear protocol π_{mult}. Let $m^{\prime c}_{\overline{\mathcal{S}},\mathcal{S}}$ be the set of messages sent by the servers in $\overline{\mathcal{S}}$ to the servers*

in $\mathcal{S}$ that are retrieved from the truncated view $u'_{\mathcal{S}\cup\mathcal{C}}$ during the execution of π_{mult}. Also, since Adv *is deterministic given $v'_{\mathcal{S}\cup\mathcal{C}}$, it is the case that $u'_{\mathcal{S}\cup\mathcal{C}}$ contains also the randomness $r'^{c}_{\mathcal{S}}$ of the servers in $\mathcal{S}$ used in π_{mult}. In addition, let $\mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{mult}}}$ be the output function of $\overline{\mathcal{S}}$ in π_{mult} as defined in Definition 5.9. The simulation proceeds as follows.*

(a) *Obtain from* Adv *the messages $\widetilde{m}'^{c}_{\mathcal{S},\overline{\mathcal{S}}}$ sent by the adversary to the servers in $\overline{\mathcal{S}}$ during the execution of π_{mult}.*

(b) *Let $\vec{h}'^{c}_{\mathcal{S}}$ be the outputs of π_{setup} corresponding to g^c that where computed in Step 2a. Simulate the honest behavior of the servers in $\mathcal{S}$ on main inputs $((g'^{a}_i, g'^{b}_i))_{S_i\in\mathcal{S}}$, auxiliary inputs $\vec{h}'^{c}_{\mathcal{S}}$, randomness $r'^{c}_{\mathcal{S}}$ and incoming messages $m'^{c}_{\overline{\mathcal{S}},\mathcal{S}}$ and obtain the messages $m'^{c}_{\mathcal{S},\overline{\mathcal{S}}}$ that should have been sent by the servers in $\mathcal{S}$ to the servers in $\overline{\mathcal{S}}$ during the execution of π_{mult}. In addition, for every server $S_i \in \mathcal{S}$ obtain the share $\overline{g}'^{c}_i$ that is the output of S_i at the after executing π_{mult}.*

(c) *Compute $\delta^c_{\overline{\mathcal{S}}} \leftarrow \mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{mult}}}(0, \gamma^c_{\overline{\mathcal{S}}}, \widetilde{m}'^{c}_{\mathcal{S},\overline{\mathcal{S}}} - m'^{c}_{\mathcal{S},\overline{\mathcal{S}}})$ where $\gamma^c_{\overline{\mathcal{S}}}$ are the part of $\gamma_{\overline{\mathcal{S}}}$ corresponding to the gate g^c (since g^c is a multiplication gate, by the construction of Π such $\gamma^c_{\overline{\mathcal{S}}}$ exists).*

(d) *The simulator now determines further entries of the additive attack A on the circuit C. Notice that $\delta^c_{\overline{\mathcal{S}}}$ is a vector of $t+1$ shares of $\mathcal{SS}$ and thus, since $\mathcal{SS}$ is dense, this forms a valid sharing of some value.*

Compute $\alpha^c \leftarrow \mathsf{recover}(\delta^c_{\overline{\mathcal{S}}}, \overline{\mathcal{S}})$ and every gate g^d connected to g^c set $A_{c,d} \leftarrow \alpha^c$. Finally, since $\mathcal{SS}$ is redundant, the simulator is able to compute the shares $(\delta^c_i)_{S_i\in\mathcal{S}}$ for the servers in $\mathcal{S}$ that are compatible with $\delta^c_{\overline{\mathcal{S}}}$.

(e) *Next, the simulator proceeds to compute the shares $(g'^{c}_i)_{S_i\in\mathcal{S}}$. For every $S_i \in \mathcal{S}$ compute $g'^{c}_i \leftarrow \overline{g}'^{c}_i + \delta^c_i$ and saves it for later use.*

6. **Output recovery phase.** *At the end of the of the circuit evaluation phase, for each output gate g^z each corrupted server S_i holds a share $\widetilde{g}^z_i$ of the supposed output. Recall that only C_1 is learn the output. Thus, there are two cases to consider based on whether C_1 is corrupted or not:*

- C_1 is corrupted. *In this case only the adversary learns the output. The simulation proceeds as follows:*
  (a) *The simulator sets to 0 all the coordinates of A that were not previously set.*
  (b) *The simulator invokes the trusted party computing $\widetilde{f}_C$ with the inputs of the corrupted parties and with the aforementioned wire corruptions A. The trusted party responds to the simulator with the outputs y for C_1.*
  (c) *For each output gate g^z of C that is connected to an output of some gate g^i the simulator chooses shares of y_z that are compatible with $(g^i_j)_{S_j\in\mathcal{S}}$, adds them to $u'_{\mathcal{S}\cup\mathcal{C}}$ and sends them to* Adv *(notice that according to Corollary 5.1 this could always be done).*
  (d) *The simulator outputs $u'_{\mathcal{S}\cup\mathcal{C}}$.*
- C_1 is honest. *Denote by $\mathcal{S}_{t+1} = \{S_1, \cdots, S_{t+1}\} \cap \mathcal{S}$. The simulation proceeds as follows:*
  (a) *For each output gate g^z of C as above and for each corrupted server $S_j \in \mathcal{S}_{t+1}$ obtain the shares $\widetilde{g}^z_j$ that S_j sends to C_1.*
  (b) *For each output gate g^z of C, the simulator sets $(A_{\mathsf{out}})_i \leftarrow \sum_{S_j\in\mathcal{S}_{t+1}} a_j(\widetilde{g}^z_j - g'^{z}_j)$ where the constants $(a_j)_{S_j\in\mathcal{S}_{t+1}}$ are used in* recover *to recover the values from shares held by the servers in $\{S_1, \cdots, S_{t+1}\}$.*
  (c) *The simulator sets to 0 all the coordinates of A that were not previously set and invokes the trusted party computing $\widetilde{f}_C$ with the inputs of the corrupted parties and with the wire corruptions A computed so far.*

(d) The simulator outputs $u'_{\mathcal{S}\cup\mathcal{C}}$.

The following Lemma claims that for any adversary Adv and for any input $\vec{x}$, the simulator Sim constructed in Construction 5.2 correctly simulates the truncated view of Adv together with the output of the honest parties during a real execution of Π on inputs x in the presence of Adv.

Lemma 5.3. *Let Π be a protocol for computing an m-client circuit $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_m} \to \mathbb{F}^{O_1}$ using $n = 2t+1$ servers that is linear-based with respect to some redundant dense linear secret sharing scheme $\mathcal{SS}$ and weakly-private against active adversaries controlling at most t servers. Then, for any adversary Adv in the real world controlling a set of clients $\mathcal{C}$ and a set $\mathcal{S}$ of servers such that $|\mathcal{S}| = t$ it holds that for any $\vec{x}$*

$$\mathsf{Ideal}_{\widetilde{f}_C,\mathsf{Sim},\mathcal{S}\cup\mathcal{C}}(\vec{x}) \equiv \mathsf{Real}_{\Pi,\mathsf{Adv},\mathcal{S}\cup\mathcal{C}}(\vec{x})$$

where $\mathsf{Sim}_{\mathsf{Adv}}$ is the simulator constructed in Construction 5.3.

Proof. Assume without loss of generality that Adv is deterministic and fix an input $\vec{x}$ for all the clients. Notice that Sim invokes $\mathsf{Sim}_{\mathsf{trunc-view}}$ in order to generate $U'_{\mathcal{C}\cup\mathcal{S}}$. By Definition 5.11 we have that $\mathsf{Sim}_{\mathsf{trunc-view}}$ correctly simulates the truncated view of Adv. Thus, Sim correctly simulates the truncated view of Adv and therefore we have that $U_{\mathcal{C}\cup\mathcal{S}} \equiv U'_{\mathcal{C}\cup\mathcal{S}}$.

It remains to prove that Sim correctly simulates the remaining rounds. For any truncated view u from the support of $U_{\mathcal{C}\cup\mathcal{S}}$ denote by $(L^u_{\overline{\mathcal{S}},\mathcal{C}}, O^u_{\overline{\mathcal{C}}})$ the random variables $(L_{\overline{\mathcal{S}},\mathcal{C}}, O_{\overline{\mathcal{C}}})$ conditioned on $U_{\mathcal{C}\cup\mathcal{S}} = u$ during a real execution of the protocol. Similarly, denote by $(L'^u_{\overline{\mathcal{S}},\mathcal{C}}, O'^u_{\overline{\mathcal{C}}})$ the random variables $(L'_{\overline{\mathcal{S}},\mathcal{C}}, O'_{\overline{\mathcal{C}}})$ conditioned on $U'_{\mathcal{C}\cup\mathcal{S}} = u$ during an ideal execution of the protocol. Since $\mathsf{Real}_{\Pi,\mathsf{Adv},\mathcal{S}\cup\mathcal{C}}(\vec{x}) = (U_{\mathcal{C}\cup\mathcal{S}}, L_{\overline{\mathcal{S}},\mathcal{C}}, O_{\overline{\mathcal{C}}})$ and $\mathsf{Ideal}_{\widetilde{f}_C,\mathsf{Sim},\mathcal{S}\cup\mathcal{C}}(\vec{x}) = (U'_{\mathcal{C}\cup\mathcal{S}}, L'_{\overline{\mathcal{S}},\mathcal{C}}, O'_{\overline{\mathcal{C}}})$ it is therefore sufficient to prove that for any truncated view u it holds that $(L^u_{\overline{\mathcal{S}},\mathcal{C}}, O^u_{\overline{\mathcal{C}}}) \equiv (L'^u_{\overline{\mathcal{S}},\mathcal{C}}, O'^u_{\overline{\mathcal{C}}})$.

We now proceed to analyze the setup phase of Π. For any truncated view u from the support of $U_{\mathcal{C}\cup\mathcal{S}}$ let $H_{\overline{\mathcal{S}},u}$ be the random variables corresponding to the outputs of the servers in $\overline{\mathcal{S}}$ after an honest execution of π_{setup} conditioned on $U_{\mathcal{C}\cup\mathcal{S}} = u$. Similarly let $\widetilde{H}_{\overline{\mathcal{S}},u}$ the random variables corresponding to the outputs of the servers in $\overline{\mathcal{C}}$ after a real execution of π_{setup} conditioned on $U_{\mathcal{C}\cup\mathcal{S}} = u$.

Finally, since Adv is deterministic, any truncated view u uniquely determines the values the vector $\gamma^{\pi_{\mathsf{setup}}}_{\overline{\mathcal{S}},u}$ as computed in Step 2c of Sim.

We now claim that even an active adversary, cannot effect the execution of the preprocessing phase of a linear protocol beside forcing all the honest servers to add some (adversarially controlled) offset to their outputs.

Claim 5.2. *For any truncated view u in the support of $U_{\mathcal{C}\cup\mathcal{S}}$, it holds that $\widetilde{H}_{\overline{\mathcal{S}},u} \equiv H_{\overline{\mathcal{S}},u} + \gamma^{\pi_{\mathsf{setup}}}_{\overline{\mathcal{S}},u}$.*

Proof. Fix a truncated view u from the support of $U_{\mathcal{C}\cup\mathcal{S}}$. Notice that the truncated view u uniquely determines the messages $m^{\pi_{\mathsf{setup}},\mathsf{honest}}_{\mathcal{S},\overline{\mathcal{S}}}$ sent from the servers in $\mathcal{S}$ to the servers in $\overline{\mathcal{S}}$ during an honest execution of π_{setup}. In addition, since Adv is deterministic, the truncated view u also uniquely determines the messages $\widetilde{m}^{\pi_{\mathsf{setup}}}_{\mathcal{S},\overline{\mathcal{S}}}$ sent by the adversary to the servers in $\overline{\mathcal{S}}$ during a real execution of π_{setup}, the messages $m'^{\pi_{\mathsf{setup}}}_{\mathcal{S},\overline{\mathcal{S}}}$ sent to Sim in Step 2a during an ideal execution of the π_{setup} and the messages $\widetilde{m}'^{\pi_{\mathsf{setup}}}_{\mathcal{S},\overline{\mathcal{S}}}$ sent to Sim in Step 2b during an ideal execution of the π_{setup}. By construction we have that $m'^{\pi_{\mathsf{setup}}}_{\mathcal{S},\overline{\mathcal{S}}} = m^{\pi_{\mathsf{setup}},\mathsf{honest}}_{\mathcal{S},\overline{\mathcal{S}}}$ and that $\widetilde{m}'^{\pi_{\mathsf{setup}}}_{\mathcal{S},\overline{\mathcal{S}}} = \widetilde{m}^{\pi_{\mathsf{setup}}}_{\mathcal{S},\overline{\mathcal{S}}}$.

Recall that π_{setup} is a linear protocol and let $M^{\pi_{\mathsf{setup}}}_{\mathsf{inp},\overline{\mathcal{S}},u}$ be the random variables corresponding to the input dependent messages sent by the servers in $\overline{\mathcal{S}}$ to themselves during a real execution of π_{setup}.

In addition, let be the random variables $M^{\pi_{\mathsf{setup}}}_{\mathsf{inp},\overline{\mathcal{S}}}$ conditioned on $U_{\mathcal{C}\cup\mathcal{S}} = u$. Since π_{setup} is a linear protocol for any m_1, m_1', m_2, m_2' it holds that

$$\mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{setup}}}(m_1 + m_1', \perp, m_2 + m_2') = \mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{setup}}}(m_1, \perp, m_2) + \mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{setup}}}(m_1', \perp, m_2').$$

Thus we have that,

$$\begin{aligned}
\widetilde{H}_{\overline{\mathcal{S}},u} &= \mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{setup}}}\left(M^{\pi_{\mathsf{setup}}}_{\mathsf{inp},\overline{\mathcal{S}},u}, \perp, \widetilde{m}^{\pi_{\mathsf{setup}}}_{\mathcal{S},\overline{\mathcal{S}}}\right) \\
&= \mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{setup}}}\left(M^{\pi_{\mathsf{setup}}}_{\mathsf{inp},\overline{\mathcal{S}},u}, \perp, m^{\pi_{\mathsf{setup}},\mathsf{honest}}_{\mathcal{S},\overline{\mathcal{S}}} + \widetilde{m}^{\pi_{\mathsf{setup}}}_{\mathcal{S},\overline{\mathcal{S}}} - m^{\pi_{\mathsf{setup}},\mathsf{honest}}_{\mathcal{S},\overline{\mathcal{S}}}\right) \\
&= \mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{setup}}}\left(M^{\pi_{\mathsf{setup}}}_{\mathsf{inp},\overline{\mathcal{S}},u}, \perp, m^{\pi_{\mathsf{setup}},\mathsf{honest}}_{\mathcal{S},\overline{\mathcal{S}}}\right) + \mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{setup}}}\left(0, \perp, \widetilde{m}^{\pi_{\mathsf{setup}}}_{\mathcal{S},\overline{\mathcal{S}}} - m^{\pi_{\mathsf{setup}},\mathsf{honest}}_{\mathcal{S},\overline{\mathcal{S}}}\right).
\end{aligned}$$

By construction of π_{setup} we have that $\mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{setup}}}(M^{\pi_{\mathsf{setup}}}_{\mathsf{inp},\overline{\mathcal{S}},u}, \perp, m^{\pi_{\mathsf{setup}},\mathsf{honest}}_{\mathcal{S},\overline{\mathcal{S}}}) \equiv H_{\overline{\mathcal{S}},u}$ and by construction of $\gamma^{\pi_{\mathsf{setup}}}_{\overline{\mathcal{S}},u}$ it holds that

$$\gamma^{\pi_{\mathsf{setup}}}_{\overline{\mathcal{S}},u} = \mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{setup}}}(0, \perp, \widetilde{m}'^{\pi_{\mathsf{setup}}}_{\mathcal{S},\overline{\mathcal{S}}} - m'^{\pi_{\mathsf{setup}}}_{\mathcal{S},\overline{\mathcal{S}}}) = \mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{setup}}}(0, \perp, \widetilde{m}^{\pi_{\mathsf{setup}}}_{\mathcal{S},\overline{\mathcal{S}}} - m^{\pi_{\mathsf{setup}},\mathsf{honest}}_{\mathcal{S},\overline{\mathcal{S}}}).$$

Thus we have that,

$$\mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{setup}}}\left(M^{\pi_{\mathsf{setup}}}_{\mathsf{inp},\overline{\mathcal{S}},u}, \perp, m^{\pi_{\mathsf{setup}},\mathsf{honest}}_{\mathcal{S},\overline{\mathcal{S}}}\right) + \mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{setup}}}\left(0, \perp, \widetilde{m}^{\pi_{\mathsf{setup}}}_{\mathcal{S},\overline{\mathcal{S}}} - m^{\pi_{\mathsf{setup}},\mathsf{honest}}_{\mathcal{S},\overline{\mathcal{S}}}\right) \equiv H_{\overline{\mathcal{S}},u} + \gamma^{\pi_{\mathsf{setup}}}_{\overline{\mathcal{S}},u}.$$

■

We now proceed to analyze the randomness generation phase of Π. For any truncated view u in the support of $U_{\mathcal{C}\cup\mathcal{S}}$ denote by $\widetilde{Y}^{\mathsf{rand}}_{\overline{\mathcal{S}},u}$ the shares held by the honest servers corresponding to the output of π_{rand} during a real execution of π_{rand} conditioned on $U_{\mathcal{C}\cup\mathcal{S}} = u$. Similarly, denote by $Y^{\mathsf{rand}}_{\overline{\mathcal{S}},u}$ the shares held by the honest servers corresponding to the output of π_{rand} during an honest execution of π_{rand} conditioned on $U'_{\mathcal{C}\cup\mathcal{S}} = u$. Finally, since Adv is deterministic, any truncated view u uniquely determines the values a vector $\gamma^{\pi_{\mathsf{rand}}}_{\overline{\mathcal{S}},u}$ as computed in Step 3c of Sim.

The proof of the following claim is similar to the proof of Claim 5.2 and is omitted.

Claim 5.3. *For any truncated view u in the support of $U_{\mathcal{C}\cup\mathcal{S}}$, it holds that $\widetilde{Y}^{\mathsf{rand}}_{\overline{\mathcal{S}},u} \equiv Y^{\mathsf{rand}}_{\overline{\mathcal{S}},u} + \gamma^{\pi_{\mathsf{rand}}}_{\overline{\mathcal{S}},u}$.*

We now proceed to analyze the input sharing and circuit evaluation phases of Π.

Let $g^1, \cdots, g^{|C|}$ be a topological ordering of the gates of C starting from the input gates and ending with the output gates. For any truncated view u in the support of $U_{\mathcal{C}\cup\mathcal{S}}$ and for any gate $g^c \in C$ with input wires that are the outputs of gates g^a, g^b we denote by $\widetilde{Y}^c_{\overline{\mathcal{S}},u}$ the shares held by the honest servers corresponding to the output of g^c during a real execution of the protocol conditioned on $U_{\mathcal{C}\cup\mathcal{S}} = u$.

In addition, for any $1 \leq c \leq |C|$, and for any truncated view u in the support of $U'_{\mathcal{C}\cup\mathcal{S}}$ denote by Z^c_u the random variable corresponding to the value of the output wire of g^c during an ideal execution of the protocol (as computed by $\widetilde{f}_C$) conditioned on $U'_{\mathcal{C}\cup\mathcal{S}} = u$. Also, for any truncated view u in the support of $U'_{\mathcal{C}\cup\mathcal{S}}$ denote by A^u the vector $A \in \mathbb{F}^w$ computed by the simulator conditioned on $U'_{\mathcal{C}\cup\mathcal{S}} = u$. Notice that A^u is completely determined by the the truncated view u and the deterministic adversary Adv. By construction, for a gate g^a and for any pair of non-output gates $g^b, g^{b'}$ connected to the output of g^a it holds that $A^u_{a,b} = A^u_{a,b'}$. Thus, we denote by A^a_u the value $A_{a,b}$ for every gate g^b connected to the output of g^a.

Moreover, for any truncated view u in the support of $U'_{\mathcal{C}\cup\mathcal{S}}$ and for any gate g^c denote by $y^c_{\mathcal{S},u}$ the variables $(g'^c_k)_{S_k\in\mathcal{S}}$ generated in Steps 3(d)ii, 4(a)i, 4(b)iii, 5e and during the processing of addition and subtraction gates of the simulator. Intuitively, these variables represent shares of the output of g^c that the adversary should have if it where to start behaving honestly during the evaluation of the output of g^c. Notice that these shares are completely determined by u and the deterministic adversary Adv.

Finally, denote by $\mathsf{recover}_T\left(s_{i_1},\cdots,s_{i_{|T|}}\right)$ the result of recovering the value from the shares $(s_{i_1},\cdots,s_{i_T})$ held by the parties in T if $s_{i_1},\cdots,s_{i_T}$ are a valid sharing and $\perp$ otherwise.

The following claim states that the shares $(\widetilde{Y}^1_{\overline{\mathcal{S}},u},\cdots,\widetilde{Y}^c_{\overline{\mathcal{S}},u})$ held by the honest servers after a real execution of the protocol in the presence of Adv correspond to the values $(Z^1_u+A^1_u,\cdots,Z^{|C|}_u+A^{|C|}_u)$ of the internal wires inside $\widetilde{C}$ when attacked by the additive attack A. Moreover, the shares $\left(y^1_{\mathcal{S},u},\cdots,y^{|C|}_{\mathcal{S},u}\right)$ as computed by the simulator are consistent with the shares held by the honest servers.

Claim 5.4. *For any truncated view u in the support of $U_{\mathcal{C}\cup\mathcal{S}}$, it holds that*

$$\left(Z^1_u+A^1_u,\cdots,Z^{|C|}_u+A^{|C|}_u\right)\equiv\left(\mathsf{recover}_{S_1,\cdots,S_n}((\widetilde{Y}^1_{\overline{\mathcal{S}},u},y^1_{\mathcal{S},u})),\cdots,\mathsf{recover}_{S_1,\cdots,S_n}((\widetilde{Y}^{|C|}_{\overline{\mathcal{S}},u},y^{|C|}_{\mathcal{S},u}))\right).$$

Proof. Fix a truncated view u from the support of $U_{\mathcal{C}\cup\mathcal{S}}$ we now proceed to analyze the input sharing and circuit evaluation phases of Sim conditioned on $U_{\mathcal{C}\cup\mathcal{S}}=u$.

The proof is by induction on the structure of C starting from the input gated and proceeding to the output gates.

Basis. If g^c is an input gate belongs to a honest client then the claim is immediate since $A^c=0$. Similarly, if g^c is an input gate belonging to a corrupted client C, let x^c be the input of C to g^c by construction of A^c we have that

$$\mathsf{recover}_{S_1,\cdots,S_n}((\widetilde{Y}^c_{\overline{\mathcal{S}},u},y^c_{\mathcal{S},u}))\equiv x^c+A^c_u=Z^c_u+A^c_u.$$

Induction hypothesis. Assume that for any $1\leq c\leq|C|$ it holds that

$$\left(Z^1_u+A^1_u,\cdots,Z^{c-1}_u+A^{c-1}_u\right)\equiv\left(\mathsf{recover}_{S_1,\cdots,S_n}((\widetilde{Y}^1_{\overline{\mathcal{S}},u},y^1_{\mathcal{S},u})),\cdots,\mathsf{recover}_{S_1,\cdots,S_n}((\widetilde{Y}^{c-1}_{\overline{\mathcal{S}},u},y^{c-1}_{\mathcal{S},u}))\right).$$

Induction step. Let g^c be a gate inside C with inputs g^a, g^b such that $c>a$ and $c>b$. Assume without loss of generality that $b>a$ and fix values $\left(\widetilde{y}^1_{\overline{\mathcal{S}},u},\cdots,\widetilde{y}^b_{\overline{\mathcal{S}},u}\right)$ from the support of $\left(\widetilde{Y}^1_{\overline{\mathcal{S}},u},\cdots,\widetilde{Y}^b_{\overline{\mathcal{S}},u}\right)$ and values $\left(z^1_u,\cdots,z^b_u\right)$ from the support of $\left(Z^1_u,\cdots,Z^b_u\right)$. We now claim that conditioned on the above selection it holds that

$$Z^c_u+A^c_u\equiv\mathsf{recover}_{S_1,\cdots,S_n}((\widetilde{Y}^c_{\overline{\mathcal{S}},u},y^c_{\mathcal{S},u})).$$

From the induction hypothesis we have that,

$$z^a_u+A^a_u\equiv\mathsf{recover}_{S_1,\cdots,S_n}((\widetilde{y}^a_{\overline{\mathcal{S}},u},y^a_{\mathcal{S},u}))\text{ and }\ z^b_u+A^b_u\equiv\mathsf{recover}_{S_1,\cdots,S_n}((\widetilde{y}^b_{\overline{\mathcal{S}},u},y^b_{\mathcal{S},u})).$$

Notice that the random variable Z^c_u conditioned on the above selection of $\left(\widetilde{y}^1_{\overline{\mathcal{S}},u},\cdots,\widetilde{y}^b_{\overline{\mathcal{S}},u}\right)$ and $\left(z^1_u,\cdots,z^b_u\right)$ is uniquely determined by the fixed $z^a_u+A^a_u$ and $z^b_u+A^b_u$. We split the proof into three cases.

Handling addition and subtraction gates. If g^c is an addition gate, notice that $A_u^c = 0$. Thus, by the linearity of $\mathcal{SS}$ we have that

$$\begin{aligned} Z_u^c + A_u^c &= Z_u^c \\ &= z_u^a + A_u^a + z_u^b + A_u^b \\ &\equiv \mathsf{recover}_{S_1,\cdots,S_n}\left((\widetilde{y}^a_{\overline{\mathcal{S}},u}, y^a_{\mathcal{S},u})\right) + \mathsf{recover}_{S_1,\cdots,S_n}\left((\widetilde{y}^b_{\overline{\mathcal{S}},u}, y^b_{\mathcal{S},u})\right) \\ &= \mathsf{recover}_{S_1,\cdots,S_n}\left((\widetilde{y}^a_{\overline{\mathcal{S}},u}, y^a_{\mathcal{S},u}) + (\widetilde{y}^b_{\overline{\mathcal{S}},u}, y^b_{\mathcal{S},u})\right) \\ &= \mathsf{recover}_{S_1,\cdots,S_n}\left((\widetilde{Y}^c_{\overline{\mathcal{S}},u}, y^c_{\mathcal{S},u})\right) \end{aligned}$$

where the last transition follows from the Constructions of Π and Sim. The handling of subtraction gates is similar and therefore is omitted.

Handling randomness gates. If g^c is a randomness gate, by construction of Π we have that $\mathsf{recover}_{\overline{\mathcal{S}}}(Y^c_{\overline{\mathcal{S}},u}) \equiv Z_u^c$ where $Y^c_{\overline{\mathcal{S}},u}$ is the restriction of $Y^{\mathsf{rand}}_{\overline{\mathcal{S}},u}$ to the shares corresponding to g^c. Let $\gamma^c_{\overline{\mathcal{S}},u}$ be the restriction of $\gamma^{\pi_{\mathsf{rand}}}_{\overline{\mathcal{S}},u}$ to the entries corresponding to g^c. By construction we have that $A_u^c = \mathsf{recover}_{\overline{\mathcal{S}}}(\gamma^c_{\overline{\mathcal{S}},u})$. Thus, using Claim 5.3 and the linearity of $\mathcal{SS}$ we have that

$$\begin{aligned} \mathsf{recover}_{\overline{\mathcal{S}}}(\widetilde{Y}^c_{\overline{\mathcal{S}},u}) &\equiv \mathsf{recover}_{\overline{\mathcal{S}}}(Y^c_{\overline{\mathcal{S}},u} + \gamma^c_{\overline{\mathcal{S}},u}) \\ &= \mathsf{recover}_{\overline{\mathcal{S}}}(Y^c_{\overline{\mathcal{S}},u}) + \mathsf{recover}_{\overline{\mathcal{S}}}(\gamma^c_{\overline{\mathcal{S}},u}) \\ &\equiv Z_u^c + A_u^c. \end{aligned}$$

Denote by $\overline{y}^c_{\mathcal{S},u} = (g_i^c)_{S_i \in \mathcal{S}}$ the shares obtained during Step 3a of the simulator. Notice that these shares are completely determined by the truncated view u and by Adv. Finally, notice that by construction of π_{rand} we have that

$$\mathsf{recover}_{S_1,\cdots,S_n}\left((Y^c_{\overline{\mathcal{S}},u}, \overline{y}^c_{\mathcal{S},u})\right) \equiv Z_u^c.$$

Thus, by construction of $\overline{y}^c_{\mathcal{S},u}$ we have that

$$\mathsf{recover}_{S_1,\cdots,S_n}\left((\widetilde{Y}^c_{\overline{\mathcal{S}},u}, y^c_{\mathcal{S},u})\right) \equiv Z_u^c + A_u^c.$$

Handling multiplication gates. If g^c is a multiplication gate, then notice that by the definition of $\widetilde{f}_C$ and by the induction hypothesis it holds that

$$\begin{aligned} Z_u^c &= (z_u^a + A_u^a)\left(z_u^b + A_u^b\right) \\ &\equiv \left(\mathsf{recover}_{S_1,\cdots,S_n}((\widetilde{y}^a_{\overline{\mathcal{S}},u}, y^a_{\mathcal{S},u}))\right)\left(\mathsf{recover}_{S_1,\cdots,S_n}((\widetilde{y}^b_{\overline{\mathcal{S}},u}, y^b_{\mathcal{S},u}))\right). \end{aligned}$$

Next, denote by $\widetilde{M}'^c_{\mathcal{S},\overline{\mathcal{S}}}$ the messages obtained by the simulator in Step 5a and denote by $\widetilde{M}^c_{\mathcal{S},\overline{\mathcal{S}}}$ the messages obtained by the honest servers corresponding to the output of g^c during a real execution of the protocol. Since we have that $U'_{\mathcal{C}\cup\mathcal{S}} \equiv U_{\mathcal{C}\cup\mathcal{S}}$ it holds that $(U'_{\mathcal{C}\cup\mathcal{S}}, \widetilde{M}'^c_{\mathcal{S},\overline{\mathcal{S}}}) \equiv (U_{\mathcal{C}\cup\mathcal{S}}, \widetilde{M}^c_{\mathcal{S},\overline{\mathcal{S}}})$.

Notice that the truncated view u and Adv completely determine both $\widetilde{M}'^c_{\mathcal{S},\overline{\mathcal{S}}}$ and $\widetilde{M}^c_{\mathcal{S},\overline{\mathcal{S}}}$ as well as the messages $m'^c_{\mathcal{S},\overline{\mathcal{S}}}$ obtained by the simulator in Step 5b. Thus, denote by $\widetilde{m}^c_{\mathcal{S},\overline{\mathcal{S}}}$ the value of both $\widetilde{M}'^c_{\mathcal{S},\overline{\mathcal{S}}}$ and $\widetilde{M}^c_{\mathcal{S},\overline{\mathcal{S}}}$ as determined by u.

For any set of parties T denote by $\mathsf{inpMsg}_{T,\pi_{\mathsf{mult}}}(x_T)$ the nondeterministic functionality that computes the values of the input-dependent messages that the parties in T send to all the parties given their deterministic inputs x_T and using fresh randomness. In addition, denote by $\widetilde{H}^c_{\overline{\mathcal{S}},u}$ and $\gamma^c_{\overline{\mathcal{S}},u}$ the restrictions of $\widetilde{H}_{\overline{\mathcal{S}},u}$ and $\gamma_{\overline{\mathcal{S}},u}$ respectively to the values corresponding to the gate g^c.

By Claim 5.2 we have that $\widetilde{H}^c_{\overline{\mathcal{S}},u} - \gamma^c_{\overline{\mathcal{S}},u} \equiv H^c_{\overline{\mathcal{S}},u}$ where $H^c_{\overline{\mathcal{S}},u}$ is $H_{\overline{\mathcal{S}},u}$ restricted to the values corresponding to the gate g^c. Thus, by construction of Π, π_{mult} and $m'^c_{\mathcal{S},\overline{\mathcal{S}}}$ it holds that

$$\begin{aligned}
&\mathsf{recover}_{\overline{\mathcal{S}}}\Big(\mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{mult}}}\Big(\mathsf{inpMsg}_{\mathcal{S},\pi_{\mathsf{mult}}}\left(\widetilde{y}^a_{\overline{\mathcal{S}},u}, \widetilde{y}^b_{\overline{\mathcal{S}},u}\right), \widetilde{H}^c_{\overline{\mathcal{S}},u} - \gamma^c_{\overline{\mathcal{S}},u}, m'^c_{\mathcal{S},\overline{\mathcal{S}}}\Big)\Big) \qquad (24)\\
&\qquad \equiv \mathsf{recover}_{\overline{\mathcal{S}}}\Big(\mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{mult}}}\Big(\mathsf{inpMsg}_{\mathcal{S},\pi_{\mathsf{mult}}}\left(\widetilde{y}^a_{\overline{\mathcal{S}},u}, \widetilde{y}^b_{\overline{\mathcal{S}},u}\right), H^c_{\overline{\mathcal{S}},u}, m'^c_{\mathcal{S},\overline{\mathcal{S}}}\Big)\Big)\\
&\qquad = \left(\mathsf{recover}_{S_1,\cdots,S_n}((\widetilde{y}^a_{\overline{\mathcal{S}},u}, y^a_{\mathcal{S},u}))\right)\left(\mathsf{recover}_{S_1,\cdots,S_n}((\widetilde{y}^b_{\overline{\mathcal{S}},u}, y^b_{\mathcal{S},u}))\right)\\
&\qquad \equiv (z^a_u + A^a_u)\left(z^b_u + A^b_u\right)\\
&\qquad = Z^c_u.
\end{aligned}$$

Second, notice that

$$\begin{aligned}
&\mathsf{recover}_{\overline{\mathcal{S}}}(\widetilde{Y}^c_{\overline{\mathcal{S}},u})\\
&= \mathsf{recover}_{\overline{\mathcal{S}}}\left(\mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{mult}}}\left(\mathsf{inpMsg}_{\mathcal{S},\pi_{\mathsf{mult}}}\left(\widetilde{y}^a_{\overline{\mathcal{S}},u}, \widetilde{y}^b_{\overline{\mathcal{S}},u}\right), \widetilde{h}^c_{\overline{\mathcal{S}},u}, \widetilde{m}^c_{\mathcal{S},\overline{\mathcal{S}}}\right)\right)\\
&= \mathsf{recover}_{\overline{\mathcal{S}}}\left(\mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{mult}}}\left(\mathsf{inpMsg}_{\mathcal{S},\pi_{\mathsf{mult}}}\left(\widetilde{y}^a_{\overline{\mathcal{S}},u}, \widetilde{y}^b_{\overline{\mathcal{S}},u}\right), \widetilde{h}^c_{\overline{\mathcal{S}},u} - \gamma^c_{\overline{\mathcal{S}},u} + \gamma^c_{\overline{\mathcal{S}},u}, m'^c_{\mathcal{S},\overline{\mathcal{S}}} + \widetilde{m}^c_{\mathcal{S},\overline{\mathcal{S}}} - m'^c_{\mathcal{S},\overline{\mathcal{S}}}\right)\right)\\
&= \mathsf{recover}_{\overline{\mathcal{S}}}\left(\mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{mult}}}\left(\mathsf{inpMsg}_{\mathcal{S},\pi_{\mathsf{mult}}}\left(\widetilde{y}^a_{\overline{\mathcal{S}},u}, \widetilde{y}^b_{\overline{\mathcal{S}},u}\right), \widetilde{h}^c_{\overline{\mathcal{S}},u} - \gamma^c_{\overline{\mathcal{S}},u}, m'^c_{\mathcal{S},\overline{\mathcal{S}}}\right)\right)\\
&\quad + \mathsf{recover}_{\overline{\mathcal{S}}}\left(\mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{mult}}}\left(0, \gamma^c_{\overline{\mathcal{S}},u}, \widetilde{m}^c_{\mathcal{S},\overline{\mathcal{S}}} - m'^c_{\mathcal{S},\overline{\mathcal{S}}}\right)\right)
\end{aligned}$$

where the last transition follows the linearity of $\mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{mult}}}$ and $\mathcal{SS}$. Next, notice that from Equation 24 and from the construction of A we have that,

$$\begin{aligned}
&\mathsf{recover}_{\overline{\mathcal{S}}}\Big(\mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{mult}}}\Big(\mathsf{inpMsg}_{\mathcal{S},\pi_{\mathsf{mult}}}\left(\widetilde{y}^a_{\overline{\mathcal{S}},u}, \widetilde{y}^b_{\overline{\mathcal{S}},u}\right), \widetilde{h}^c_{\overline{\mathcal{S}},u} - \gamma^c_{\overline{\mathcal{S}},u}, m'^c_{\mathcal{S},\overline{\mathcal{S}}}\Big)\Big)\\
&\qquad + \mathsf{recover}_{\overline{\mathcal{S}}}\left(\mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{mult}}}\left(0, \gamma^c_{\overline{\mathcal{S}},u}, \widetilde{m}^c_{\mathcal{S},\overline{\mathcal{S}}} - m'^c_{\mathcal{S},\overline{\mathcal{S}}}\right)\right)\\
&\qquad \equiv Z^c_u + \mathsf{recover}_{\overline{\mathcal{S}}}\left(\mathsf{output}_{\overline{\mathcal{S}},\pi_{\mathsf{mult}}}\left(0, \gamma^c_{\overline{\mathcal{S}},u}, \widetilde{m}^c_{\mathcal{S},\overline{\mathcal{S}}} - m'^c_{\mathcal{S},\overline{\mathcal{S}}}\right)\right)\\
&\qquad = Z^c_u + A^c_u
\end{aligned}$$

Thus, we have proved that $\mathsf{recover}_{\overline{\mathcal{S}}}(\widetilde{Y}^c_{\overline{\mathcal{S}},u}) \equiv Z^c_u + A^c_u$. Denote by $\overline{y}^c_{\mathcal{S},u} = (g^c_i)_{S_i \in \mathcal{S}}$ the shares obtained during Step 5b of the simulator. Notice that these shares are completely determined by the truncated view u and by Adv. In addition, let δ^c be the values computed in Steps 5c and 5d of the simulator. Notice that these values are also completely determined by the truncated view u and by Adv. Finally, let $\delta^c_{\mathcal{S}}$ and $\delta^c_{\overline{\mathcal{S}}}$ be the restrictions of δ^c to the parties in $\mathcal{S}$ and $\overline{\mathcal{S}}$ respectively. Notice that by construction we have that $\mathsf{recover}_{S_1,\cdots,S_n}(\delta^c) = A^c_u$ and thus by the linearity of $\mathcal{SS}$ it holds that

$$\mathsf{recover}_{\overline{\mathcal{S}}}\left(\widetilde{Y}^c_{\overline{\mathcal{S}},u} - \delta^c_{\overline{\mathcal{S}}}\right) \equiv Z^c_u.$$

Next, notice that since $\mathsf{recover}_{S_1,\cdots,S_n}((\widetilde{y}^a_{\overline{\mathcal{S}},u}, y^a_{\mathcal{S},u})) \neq \perp$ and since $\mathsf{recover}_{S_1,\cdots,S_n}((\widetilde{y}^b_{\overline{\mathcal{S}},u}, y^b_{\mathcal{S},u})) \neq \perp$ by the construction of $\overline{y}^c_{\mathcal{S}}$ we have that

$$\mathsf{recover}_{S_1,\cdots,S_n}\left(\widetilde{Y}^c_{\overline{\mathcal{S}},u} - \delta^c_{\overline{\mathcal{S}}}, \overline{y}^c_{\mathcal{S},u}\right) = \mathsf{recover}_{\overline{\mathcal{S}}}\left(\widetilde{Y}^c_{\overline{\mathcal{S}},u} - \delta^c_{\overline{\mathcal{S}}}\right) \equiv Z^c_u.$$

Finally, since by construction we have that $\overline{y}^c_{\mathcal{S},u} + \delta^c_{\mathcal{S}} = y^c_{\mathcal{S},u}$ by the linearity of $\mathcal{SS}$ we obtain

$$\begin{aligned}
\mathsf{recover}_{S_1,\cdots,S_n}\left(\widetilde{Y}^c_{\overline{\mathcal{S}},u}, y^c_{\mathcal{S},u}\right) &= \mathsf{recover}_{S_1,\cdots,S_n}\left(\widetilde{Y}^c_{\overline{\mathcal{S}},u}, y^c_{\mathcal{S},u}\right) - \mathsf{recover}_{S_1,\cdots,S_n}\left(\delta^c\right) + \mathsf{recover}_{S_1,\cdots,S_n}\left(\delta^c\right) \\
&= \mathsf{recover}_{S_1,\cdots,S_n}\left(\widetilde{Y}^c_{\overline{\mathcal{S}},u} - \delta^c_{\overline{\mathcal{S}}}, y^c_{\mathcal{S},u} - \delta^c_{\mathcal{S}}\right) + \mathsf{recover}_{S_1,\cdots,S_n}\left(\delta^c\right) \\
&= \mathsf{recover}_{S_1,\cdots,S_n}\left(\widetilde{Y}^c_{\overline{\mathcal{S}},u} - \delta^c_{\overline{\mathcal{S}}}, \overline{y}^c_{\mathcal{S},u}\right) + \mathsf{recover}_{S_1,\cdots,S_n}\left(\delta^c\right) \\
&\equiv Z^c_u + A^c_u.
\end{aligned}$$

Thus we have proved that $\mathsf{recover}_{S_1,\cdots,S_n}((\widetilde{Y}^c_{\overline{\mathcal{S}},u}, y^c_{\mathcal{S},u})) = \mathsf{recover}_{\overline{\mathcal{S}}}(\widetilde{Y}^c_{\overline{\mathcal{S}},u}) \equiv Z^c_u + A^c_u$. Therefore, for any $1 \leq c \leq |C|$ it holds that

$$\left(Z^1_u + A^1_u, \cdots, Z^c_u + A^c_u\right) \equiv \left(\mathsf{recover}_{S_1,\cdots,S_n}((\widetilde{Y}^1_{\overline{\mathcal{S}},u}, y^1_{\mathcal{S},u})), \cdots, \mathsf{recover}_{S_1,\cdots,S_n}((\widetilde{Y}^c_{\overline{\mathcal{S}},u}, y^c_{\mathcal{S},u}))\right)$$

and the claim follows. ■

We now return to the proof of Lemma 5.3. Claim 5.4 above states that the shares $(\widetilde{Y}^1_{\overline{\mathcal{S}},u}, \cdots, \widetilde{Y}^c_{\overline{\mathcal{S}},u})$ held by the honest servers after a real execution of the protocol in the presence of Adv correspond to the values $(Z^1_u + A^1_u, \cdots, Z^{|C|}_u + A^{|C|}_u)$ of the internal wires inside $\widetilde{C}$ when attacked by the additive attack $(A^1_u, \cdots, A^{|C|}_u)$. In particular, this is the case for the output gates of C. By construction of Π, the shares of an output gate will be used by some of the honest clients during the output recovery phase of Π. It therefore remains to prove that any active attack on the output recovery phase of Π can only result in yet another additive attack on the outputs of C.

Denote $H = \{S_1, \cdots, S_{t+1}\} \cap \overline{\mathcal{S}}$ and $\mathcal{S}' = \{S_1, \cdots, S_{t+1}\} \setminus H$. Next, for each gate g^c connected to an output gate g^z, let $(\widetilde{Y}^c_{H,u}, y^c_{\mathcal{S}',u})$ be the restriction of $(\widetilde{Y}^c_{\overline{\mathcal{S}},u}, y^c_{\mathcal{S},u})$ to the parties in $H \cup \mathcal{S}'$. Let $g^{c_1}, \cdots, g^{c_k}$ be the output gates of C such that g^{c_i} is i-th output of C. We have two cases to consider.

1. $\boldsymbol{C_1}$ **is corrupted.** In this case only the adversary is to learn the output. Thus, we have that $\mathcal{C} = \emptyset$ and that $O^u_{\mathcal{C}} \equiv O'^u_{\mathcal{C}} \equiv \perp$. It therefore sufficient to prove that $L^u_{\overline{\mathcal{S}},\mathcal{C}} \equiv L'^u_{\overline{\mathcal{S}},\mathcal{C}}$. In the real world execution of the protocol, we have that $L^u_{\overline{\mathcal{S}},\mathcal{C}} = \left(\widetilde{Y}^{c_1}_{H,u}, \cdots, \widetilde{Y}^{c_k}_{H,u}\right)$. Notice that,

$$\begin{aligned}
&\left(\mathsf{recover}_{H\cup\mathcal{S}'}((\widetilde{Y}^{c_1}_{H,u}, y^{c_1}_{\mathcal{S}',u})), \cdots, \mathsf{recover}_{H\cup\mathcal{S}'}((\widetilde{Y}^{c_k}_{H,u}, y^{c_k}_{\mathcal{S}',u}))\right) \\
&\qquad = \left(\mathsf{recover}_{S_1,\cdots,S_n}((\widetilde{Y}^{c_1}_{\overline{\mathcal{S}},u}, y^{c_1}_{\mathcal{S},u})), \cdots, \mathsf{recover}_{S_1,\cdots,S_n}((\widetilde{Y}^{c_k}_{\overline{\mathcal{S}},u}, y^{c_k}_{\mathcal{S},u}))\right) \\
&\qquad \equiv \left(Z^{c_1}_u + A^{c_1}_u, \cdots, Z^{c_k}_u + A^{c_k}_u\right)
\end{aligned}$$

That is, for every output gate g^c the distribution $L^u_{\overline{\mathcal{S}},\mathcal{C}}$ consists of shares corresponding to the parties in H of a value sampled from the distribution $Z^c_u + A^c_u$.

First, notice that $\left(Z^{c_1}_u + A^{c_1}_u, \cdots, Z^{c_k}_u + A^{c_k}_u\right)$ is the output distribution of the trusted party for this case. Second, notice that $\left(Z^{c_1}_u + A^{c_1}_u, \cdots, Z^{c_k}_u + A^{c_k}_u\right)$ is exactly the distribution of values that are shared by the simulator and the shares of which are are sent to the adversary (such that the shares sent are compatible to the shares belonging to $\mathcal{S}$ that it already knows) and thus $L^u_{\overline{\mathcal{S}},\mathcal{C}} \equiv L'^u_{\overline{\mathcal{S}},\mathcal{C}}$.

2. $\boldsymbol{C_1}$ **is honest.** In this case we have that $L^u_{\overline{\mathcal{S}},\mathcal{C}} \equiv L'^u_{\overline{\mathcal{S}},\mathcal{C}} = \perp$ and thus $V'_{\mathcal{C}\cup\mathcal{S}} = (U'_{\mathcal{C}\cup\mathcal{S}}, \perp) \equiv (U_{\mathcal{C}\cup\mathcal{S}}, \perp) = V_{\mathcal{C}\cup\mathcal{S}}$. We now prove that $(V_{\mathcal{C}\cup\mathcal{S}}, O_{\overline{\mathcal{C}}}) \equiv (V'_{\mathcal{C}\cup\mathcal{S}}, O'_{\overline{\mathcal{C}}})$. Let $O^u_{\overline{\mathcal{C}}}$ be the random variables

$O_{\overline{\mathcal{C}}}$ conditioned on $U_{\mathcal{C}\cup\mathcal{S}} = u$. Similarly, let $O'^{u}_{\overline{\mathcal{C}}}$ be the random variables $O'_{\overline{\mathcal{C}}}$ conditioned on $U_{\mathcal{C}\cup\mathcal{S}} = u$. Notice that only C_1 is to obtain output, thus it remains to prove that $O^{u}_{C_1} \equiv O'^{u}_{C_1}$.

Let $\left(\widetilde{Y}^{c_1}_{\mathcal{S}'}, \cdots, \widetilde{Y}^{c_k}_{\mathcal{S}'}\right)$ be the shares that the adversary sends to C_1 during a real execution of the protocol. Similarly, let $\left(\widetilde{Y}'^{c_1}_{\mathcal{S}'}, \cdots, \widetilde{Y}'^{c_k}_{\mathcal{S}'}\right)$ be the distribution of shares that the adversary sends to the simulator during an ideal execution of the protocol. Notice that the values sent by the adversary at this stage during the ideal and real executions of Π are completely determined by its truncated view u. Thus, since $U_{\mathcal{C}\cup\mathcal{S}} \equiv U'_{\mathcal{C}\cup\mathcal{S}}$ it holds that

$$\left(U_{\mathcal{C}\cup\mathcal{S}}, \widetilde{Y}^{c_1}_{\mathcal{S}'}, \cdots, \widetilde{Y}^{c_k}_{\mathcal{S}'}\right) \equiv \left(U'_{\mathcal{C}\cup\mathcal{S}}, \widetilde{Y}'^{c_1}_{\mathcal{S}'}, \cdots, \widetilde{Y}'^{c_k}_{\mathcal{S}'}\right).$$

Thus, let $\left(\widetilde{y}'^{c_1}_{\mathcal{S}',u}, \cdots, \widetilde{y}'^{c_k}_{\mathcal{S}',u}\right)$ be the values of $\left(\widetilde{Y}'^{c_1}_{\mathcal{S}',u}, \cdots, \widetilde{Y}'^{c_k}_{\mathcal{S}',u}\right)$ as determined by u. In a real execution of the protocol we have that

$$\begin{aligned}
O^{u}_{C_1} &= \left(\mathsf{recover}_{H\cup\mathcal{S}'}((\widetilde{Y}^{c_1}_{H,u}, \widetilde{y}'^{c_1}_{\mathcal{S}',u})), \cdots, \mathsf{recover}_{H\cup\mathcal{S}'}((\widetilde{Y}^{c_k}_{H,u}, \widetilde{y}'^{c_k}_{\mathcal{S}',u}))\right) \\
&= \left(\mathsf{recover}_{H\cup\mathcal{S}'}((\widetilde{Y}^{c_1}_{H,u}, y^{c_1}_{\mathcal{S}',u} + \widetilde{y}'^{c_1}_{\mathcal{S}',u} - y^{c_1}_{\mathcal{S}',u})), \cdots, \mathsf{recover}_{H\cup\mathcal{S}'}((\widetilde{Y}^{c_k}_{H,u}, y^{c_k}_{\mathcal{S}',u} + \widetilde{y}'^{c_k}_{\mathcal{S}',u} - y^{c_k}_{\mathcal{S}',u}))\right) \\
&= \left(\mathsf{recover}_{H\cup\mathcal{S}'}((\widetilde{Y}^{c_1}_{H,u}, y^{c_1}_{\mathcal{S}',u})), \cdots, \mathsf{recover}_{H\cup\mathcal{S}'}((\widetilde{Y}^{c_k}_{H,u}, y^{c_k}_{\mathcal{S}',u}))\right) \\
&\quad + \left(\mathsf{recover}_{H\cup\mathcal{S}'}((0, \widetilde{y}'^{c_1}_{\mathcal{S}',u} - y^{c_1}_{\mathcal{S}',u})), \cdots, \mathsf{recover}_{H\cup\mathcal{S}'}((0, \widetilde{y}'^{c_k}_{\mathcal{S}',u} - y^{c_k}_{\mathcal{S}',u}))\right) \\
&\equiv \left(Z^{c_1}_{u} + A^{c_1}_{u}, \cdots, Z^{c_k}_{u} + A^{c_k}_{u}\right) \\
&\quad + \left(\mathsf{recover}_{H\cup\mathcal{S}'}((0, \widetilde{y}'^{c_1}_{\mathcal{S}',u} - y^{c_1}_{\mathcal{S}',u})), \cdots, \mathsf{recover}_{H\cup\mathcal{S}'}((0, \widetilde{y}'^{c_k}_{\mathcal{S}',u} - y^{c_k}_{\mathcal{S}',u}))\right) \\
&= \left(Z^{c_1}_{u} + A^{c_1}_{u} + \mathsf{recover}_{H\cup\mathcal{S}'}((0, \widetilde{y}'^{c_1}_{\mathcal{S}',u} - y^{c_1}_{\mathcal{S}',u})), \cdots, Z^{c_k}_{u} + A^{c_k}_{u} + \mathsf{recover}_{H\cup\mathcal{S}'}((0, \widetilde{y}'^{c_k}_{\mathcal{S}',u} - y^{c_k}_{\mathcal{S}',u}))\right)
\end{aligned}$$

Next, notice that $\left(\mathsf{recover}_{H\cup\mathcal{S}'}(0, \widetilde{y}'^{c_1}_{\mathcal{S}',u} - y^{c_1}_{\mathcal{S}',u}), \cdots, \mathsf{recover}_{H\cup\mathcal{S}'}(0, \widetilde{y}'^{c_k}_{\mathcal{S}',u} - y^{c_k}_{\mathcal{S}',u})\right)$ is exactly the restriction A_{out} of the additive attack A on the outputs of C. Thus we have that the output of the trusted party, and the output of C_1 in the ideal world is

$$\left(Z^{c_1}_{u} + A^{c_1}_{u} + \mathsf{recover}_{H\cup\mathcal{S}'}((0, \widetilde{y}'^{c_1}_{\mathcal{S}',u} - y^{c_1}_{\mathcal{S}',u})), \cdots, Z^{c_k}_{u} + A^{c_k}_{u} + \mathsf{recover}_{H\cup\mathcal{S}'}((0, \widetilde{y}'^{c_k}_{\mathcal{S}',u} - y^{c_k}_{\mathcal{S}',u}))\right) \equiv O'^{u}_{C_1}.$$

■

5.6 MPC using linear-based protocols

In this section, we show how to utilize the observation that any protocol Π for computing a circuit C that is linear-based and weakly-private against active adversaries actually is a secure protocol for computing the additively corruptible version $\widetilde{f}_C$ of C for the purpose of computing C in the presence of an active adversary.

First, we protect the inputs and outputs of C by transforming C into another circuit C^{AMD} that gets its inputs and computes its outputs encoded in some AMD code. Second, using the results of Section 3.3 we obtain a secure version $\widehat{C}^{\mathsf{AMD}}$ of C^{AMD}. Third, we will invoke a linear-based and weakly-private against active adversaries protocol Π for computing $\widehat{C}^{\mathsf{AMD}}$.

By the properties of Π we have that any deviation from the protocol made by an active adversary corresponds to an additive attack on $\widehat{C}^{\mathsf{AMD}}$. In addition, by the security property of $\widehat{C}^{\mathsf{AMD}}$ this additive attack corresponds to an attack on the inputs and outputs of $\widehat{C}^{\mathsf{AMD}}$. Since the inputs and outputs of $\widehat{C}^{\mathsf{AMD}}$ are protected by some private AMD code, the honest parties will be able to catch these attacks

and abort the computation. See Figure 2(b) for a graphical representation of the constructions presented in this section. We begin by defining the circuit C^{AMD}.

Construction 5.3. *Let $C : \mathbb{F}^\ell \times \cdots \times \mathbb{F}^\ell \to \mathbb{F}^\ell$ be an m-client circuit. In addition, let $(\mathsf{Enc}, \mathsf{Dec})$ be an (ℓ, k, ϵ')-AMD code. We define the randomized m-client circuit $C^{\mathsf{AMD}} : \mathbb{F}^k \times \cdots \times \mathbb{F}^k \to \mathbb{F} \times \mathbb{F}^k$ that on inputs $(x_1, \ldots, x_m)$ performs the following:*

1. *For all $1 \leq i \leq m$ compute $(b_i, x_i') \leftarrow \mathsf{Dec}(x_i)$.*
2. *Compute $b \leftarrow \sum_{i=1}^m r_i b_i$ where r_i is a random field element.*
3. *Output $(b, \mathsf{Enc}(C(x_1', \cdots, x_m')) + br')$ where r' is generated uniformly at random from $\mathbb{F}^k$.*

We now proceed to describe a protocol for C in the $\widetilde{f}_{\widehat{C}^{\mathsf{AMD}}}$-hybrid model where $\widetilde{f}_{\widehat{C}^{\mathsf{AMD}}}$ is the additively corruptible version of C.

Construction 5.4. *Let $C : \mathbb{F}^\ell \times \cdots \times \mathbb{F}^\ell \to \mathbb{F}^\ell$ be an m-client circuit and let $(\mathsf{Enc}, \mathsf{Dec})$ be an $(\ell, k, \epsilon_{\mathsf{AMD}})$-AMD code. In addition, let C^{AMD} be the circuit constructed from C in Construction 5.3 using $(\mathsf{Enc}, \mathsf{Dec})$ and let $\widehat{C}^{\mathsf{AMD}}$ be an ϵ'-secure implementation of C^{AMD}. Consider the protocol π in the $\widetilde{f}_{\widehat{C}^{\mathsf{AMD}}}$-hybrid model which on inputs $(x_1, \ldots, x_m)$ proceeds as follows:*

1. *Each client C_i locally computes $\widehat{x}_i \leftarrow \mathsf{Enc}(x_i)$.*
2. *Next, each client C_i sends its encoded inputs $\widehat{x}_i$ to the ideal functionality $\widetilde{f}_{\widehat{C}^{\mathsf{AMD}}}$.*

Upon obtaining an output (b, z) from the ideal functionality, C_1 performs the following:

1. *If $b \neq 0$ then C_1 aborts.*
2. *C_1 computes $(b', z') \leftarrow \mathsf{Dec}(z)$. If $b' \neq 0$ then C_1 aborts.*
3. *Otherwise, C_1 outputs z'.*

Theorem 5.3. *For any m-client circuit $C : \mathbb{F}^\ell \times \cdots \times \mathbb{F}^\ell \to \mathbb{F}^\ell$ the protocol π as defined in Construction 5.4 ϵ-securely computes C with abort in the $\widetilde{f}_{\widehat{C}^{\mathsf{AMD}}}$-hybrid model for $\epsilon = \epsilon_{\mathsf{AMD}} + \epsilon'$.*

Proof. Let Adv be an adversary controlling a subset of clients $\mathcal{C}$. Assume without loss of generality that Adv is deterministic, we describe a simulator $\mathsf{Sim}_{\mathsf{AMD}}$ for Adv. On inputs $\vec{x}_{\mathcal{C}}$ (of the corrupted clients in $\mathcal{C}$), $\mathsf{Sim}_{\mathsf{AMD}}$ performs the following.

1. Invoke Adv on inputs $\vec{x}_{\mathcal{C}}$ and let $\vec{x}'_{\mathcal{C}}$ be the inputs sent by Adv to the $\widetilde{f}_{\widehat{C}^{\mathsf{AMD}}}$ oracle on behalf of the corrupted clients. In addition, let A be the additive attack on $\widehat{C}^{\mathsf{AMD}}$ that is also sent by Adv to the $\widetilde{f}_{\widehat{C}^{\mathsf{AMD}}}$ oracle.
2. $\mathsf{Sim}_{\mathsf{AMD}}$ computes a vector $\mathbf{a}^{\mathsf{in}}$ and a distribution $\mathcal{A}^{\mathsf{out}}$ representing the additive attack on the inputs and outputs of $\widehat{C}^{\mathsf{AMD}}$ that is equivalent to A as defined in Definition 1.1. In addition, $\mathsf{Sim}_{\mathsf{AMD}}$ samples a vector $\mathbf{a}^{\mathsf{out}}$ from $\mathcal{A}^{\mathsf{out}}$.

We split the simulation into two cases.

- If C_1 is not corrupted. We consider three sub-cases:
    1. If it holds that $\mathbf{a}^{\mathsf{in}}_{\mathcal{C}} \neq 0$ or $\mathbf{a}^{\mathsf{out}} \neq 0$ then $\mathsf{Sim}_{\mathsf{AMD}}$ aborts. For this case the simulation is complete.
    2. If it holds that $\mathbf{a}^{\mathsf{in}}_{\mathcal{C}} = \mathbf{a}^{\mathsf{out}} = 0$ then $\mathsf{Sim}_{\mathsf{AMD}}$ computes for each $C_i \in \mathcal{C}$ the values $(b_i, x_i'') \leftarrow \mathsf{Dec}(x_i' + \mathbf{a}^{\mathsf{in}}_{C_i})$.

3. If there exists a client $C_i \in \mathcal{C}$ such that $b_i \neq 0$ then $\mathsf{Sim}_{\mathsf{AMD}}$ aborts. Otherwise, $\mathsf{Sim}_{\mathsf{AMD}}$ invokes the trusted party with the decoded inputs $x''_{\mathcal{C}}$ for the corrupted clients.

- If C_1 is corrupted. In this case only the adversary gets outputs, thus $\mathsf{Sim}_{\mathsf{AMD}}$ proceeds to set the view of the adversary as follows.
  - For each $C_i \in \mathcal{C}$ compute $(b_i, x''_i) \leftarrow \mathsf{Dec}(x'_i + \mathbf{a}^{\mathsf{in}}_{C_i})$.
  - If $\mathbf{a}^{\mathsf{in}}_{\overline{\mathcal{C}}} \neq 0$ or there exists $C_i \in \mathcal{C}$ such that $b_i \neq 0$ then set the view of the adversary to be $(r', r) + \mathbf{a}^{\mathsf{out}}$ where r' is a random non-zero field element and r is generated uniformly from $\mathbb{F}^{\ell}$.
  - If $\mathbf{a}^{\mathsf{in}}_{\overline{\mathcal{C}}} = 0$ and for all $C_i \in \mathcal{C}$ it holds that $b_i = 0$ invoke the trusted party with inputs $x''_{\mathcal{C}}$ for the corrupted clients. Let z be the output of the trusted party, the view of the adversary is set to be $(0, \mathsf{Enc}(z)) + \mathbf{a}^{\mathsf{out}}$.

We claim that for all $\vec{x}$ it holds that

$$SD\left(\mathsf{Ideal}^{abort}_{C,\mathsf{Sim}_{\mathsf{AMD}},\mathcal{C}}(\vec{x}), \mathsf{Real}^{\widetilde{f}_{\widehat{C}^{\mathsf{AMD}}}}_{\pi,\mathsf{Adv},\mathcal{C}}(\vec{x})\right) \leq \epsilon_{\mathsf{AMD}} + \epsilon'.$$

Notice that all the client but C_1 do not get any messages or outputs during π, thus, simulating the view of all the corrupted client but C_1 can be done easily buy just setting their view to be their corresponding inputs. Fix an input $\vec{x}$ for π, notice that this fixes the view of all the corrupted clients but C_1 (in case it is corrupted).

We proceed to simulate the output of C_1 in case it is honest or its view in case it is corrupted. Let A be the adversary's input to $\widetilde{f}_{\widehat{C}^{\mathsf{AMD}}}$ representing the additive attack on $\widehat{C}^{\mathsf{AMD}}$ and let $\widetilde{f}$ be the randomized functionality obtained from $\widetilde{f}_{\widehat{C}^{\mathsf{AMD}}}$ by fixing the additive attack on $\widehat{C}^{\mathsf{AMD}}$ to A. By the additive-attack security property of $\widehat{C}^{\mathsf{AMD}}$ we have that

$$SD\left(\widehat{C}^{\mathsf{AMD}}((\vec{x}_{\overline{\mathcal{C}}}, \vec{x}'_{\mathcal{C}}) + \mathbf{a}^{\mathsf{in}}) + \mathcal{A}^{\mathsf{out}}, \widetilde{f}(\vec{x})\right) \leq \epsilon'$$

where $\vec{x}_{\overline{\mathcal{C}}}$ is the inputs of the honest clients and $\vec{x}'_{\mathcal{C}}$ are the inputs of the corrupted clients as provided by the adversary. We split the proof into two cases.

- If C_1 is not corrupted we have to simulate the outputs of C_1. We have two cases to consider.
  - If $\mathbf{a}^{\mathsf{in}}_{\overline{\mathcal{C}}} \neq 0$ or decoding using Dec of $\vec{x}'_{\mathcal{C}} + \mathbf{a}^{\mathsf{in}}_{\mathcal{C}}$ fails (i.e. there exists $C_i \in \mathcal{C}$ such that $(b_i, y_i) \leftarrow \mathsf{Dec}(x'_i + \mathbf{a}^{\mathsf{in}}_i)$ and $b_i \neq 0$). In this case either the adversary decided to attack the inputs of the honest clients inside $\widehat{C}^{\mathsf{AMD}}$ (which are sent encoded using Enc) or has provided with inputs $\vec{x}'_{\mathcal{C}} + \mathbf{a}^{\mathsf{in}}_{\mathcal{C}}$ for the corrupted parties that does not decode. In both cases, by the the additive robustness of $(\mathsf{Enc}, \mathsf{Dec})$ we have that

    $$SD\left(\widehat{C}^{\mathsf{AMD}}((\vec{x}_{\overline{\mathcal{C}}}, \vec{x}'_{\mathcal{C}}) + \mathbf{a}^{\mathsf{in}}) + \mathcal{A}^{\mathsf{out}}, U^* \times U_{\ell}\right) \leq \epsilon_{\mathsf{AMD}}$$

    where U^* is the uniform distribution of the non-zero elements of $\mathbb{F}$. Finally, observe that upon receiving output from such distribution C_1 immediately aborts. Thus, we have obtained the output of C_1 is $(\epsilon_{\mathsf{AMD}} + \epsilon')$-close to abort which is exactly the simulated output of C_1.
  - If $\mathbf{a}^{\mathsf{in}}_{\overline{\mathcal{C}}} = 0$ and decoding using Dec of $\vec{x}'_{\mathcal{C}} + \mathbf{a}^{\mathsf{in}}_{\mathcal{C}}$ succeeds (i.e. for all $C_i \in \mathcal{C}$ it holds that $(b_i, y_i) \leftarrow \mathsf{Dec}(x'_i + \mathbf{a}^{\mathsf{in}}_i)$ and $b_i = 0$). In this case, let $\vec{y}$ be the decoding of $(\vec{x}_{\overline{\mathcal{C}}}, \vec{x}'_{\mathcal{C}}) + \mathbf{a}^{\mathsf{in}}$. Observe that by the additive robustness property of $(\mathsf{Enc}, \mathsf{Dec})$ and of $\widehat{C}^{\mathsf{AMD}}$ we have that the output of C_1 in the real world is $(\epsilon_{\mathsf{AMD}} + \epsilon')$-close to the distribution $\mathcal{A}^{\mathsf{out}}_{C_1}$ defined as follows: $\mathcal{A}^{\mathsf{out}}_{C_1}$ is the second output of $\mathsf{Dec}(\widehat{C}^{\mathsf{AMD}}(\mathsf{Enc}(\vec{y})))$ when $\mathcal{A}^{\mathsf{out}}$ outputs 0 and $\mathcal{A}^{\mathsf{out}}_{C_1}$ outputs an abort symbol otherwise. Next, observe that second output of $\mathsf{Dec}(\widehat{C}^{\mathsf{AMD}}(\mathsf{Enc}(\vec{x})))$ is equal to $C(x)$. Finally, observe that $\mathcal{A}^{\mathsf{out}}_{C_1}$ is exactly the distribution of the simulated output of C_1.

- If C_1 is corrupted we have to simulate its view since $O_{\overline{\mathcal{C}}} \equiv O'_{\overline{\mathcal{C}}} \equiv \perp$. We have two cases to consider.
  - If $\mathbf{a}^{\mathsf{in}}_{\overline{\mathcal{C}}} \neq 0$ or decoding using Dec of $\vec{x}'_{\mathcal{C}} + \mathbf{a}^{\mathsf{in}}_{\mathcal{C}}$ fails (i.e. there exists $C_i \in \mathcal{C}$ for which it holds that $(b_i, y_i) \leftarrow \mathsf{Dec}(x'_i + \mathbf{a}^{\mathsf{in}}_i)$ and $b_i \neq 0$). In this case either the adversary decided to attack the inputs of the honest clients inside $\widehat{C}^{\mathsf{AMD}}$ (which are sent encoded using Enc) or has provided with inputs $\vec{x}'_{\mathcal{C}} + \mathbf{a}^{\mathsf{in}}_{\mathcal{C}}$ for the corrupted parties that does not decode.
    By construction we have that the view of the clients in $\mathcal{C}$ during a real execution of π is $V_{\mathcal{C}} = (x_{\mathcal{C}}, \widetilde{f}(\vec{x}))$. Using the additive-attack security property of $\widehat{C}^{\mathsf{AMD}}$ we obtain
    $$SD\left(V_{\mathcal{C}}, (x_{\mathcal{C}}, \widehat{C}^{\mathsf{AMD}}((\vec{x}_{\overline{\mathcal{C}}}, \vec{x}'_{\mathcal{C}}) + \mathbf{a}^{\mathsf{in}}) + \mathcal{A}^{\mathsf{out}})\right) \leq \epsilon'.$$
    Using the additive robustness property of $(\mathsf{Enc}, \mathsf{Dec})$, we have that
    $$SD\left(V_{\mathcal{C}}, (x_{\mathcal{C}}, U^* \times U_k + \mathcal{A}^{\mathsf{out}})\right) \leq \epsilon_{\mathsf{AMD}} + \epsilon'.$$
    Finally, notice that by construction it holds that $(x_{\mathcal{C}}, U^* \times U_k + \mathcal{A}^{\mathsf{out}}) = V'_{\mathcal{C}}$.
  - If $\mathbf{a}^{\mathsf{in}}_{\overline{\mathcal{C}}} = 0$ and decoding using Dec of $\vec{x}'_{\mathcal{C}} + \mathbf{a}^{\mathsf{in}}_{\mathcal{C}}$ succeeds (i.e. for all $C_i \in \mathcal{C}$ it holds that $(b_i, y_i) \leftarrow \mathsf{Dec}(x'_i + \mathbf{a}^{\mathsf{in}}_i)$ and $b_i = 0$). In this case, let $\vec{y}$ be the decoding of $(\vec{x}_{\overline{\mathcal{C}}}, \vec{x}'_{\mathcal{C}}) + \mathbf{a}^{\mathsf{in}}$.
    By construction we have that $V_{\mathcal{C}} \equiv (x_{\mathcal{C}}, \widetilde{f}(\vec{x}))$. Using the additive-attack security property of $\widehat{C}^{\mathsf{AMD}}$ we obtain
    $$SD\left(V_{\mathcal{C}}, (x_{\mathcal{C}}, \widehat{C}^{\mathsf{AMD}}((\vec{x}_{\overline{\mathcal{C}}}, \vec{x}'_{\mathcal{C}}) + \mathbf{a}^{\mathsf{in}}) + \mathcal{A}^{\mathsf{out}})\right) \leq \epsilon'.$$
    Observe that for this case it holds that $\widehat{C}_{\mathsf{AMD}}((\vec{x}_{\overline{\mathcal{C}}}, \vec{x}'_{\mathcal{C}}) + \mathbf{a}^{\mathsf{in}}) = (0, \mathsf{Enc}(C(\vec{y})))$. Thus, we have that
    $$SD\Big(V_{\mathcal{C}}, \big(x_{\mathcal{C}}, (0, \mathsf{Enc}(C(\vec{y})) + \mathcal{A}^{\mathsf{out}})\big)\Big) \leq \epsilon'.$$
    Finally, notice that by construction it holds that $(x_{\mathcal{C}}, (0, \mathsf{Enc}(C(\vec{y})) + \mathcal{A}^{\mathsf{out}})) \equiv V'_{\mathcal{C}}$.

■

The following corollary states that any protocol π for privately computing a circuit C that is linear-based and weakly-private against active adversaries can be transformed into a protocol π' securely computing C in the presence of active adversaries.

Corollary 5.2. *Let n, t be positive integers such that $n = 2t + 1$ and let π be a protocol for t-privately computing a circuit C, using m clients and n servers, that is linear-based with respect to some dense and redundant secret sharing scheme and is weakly-private against active adversaries controlling at most t servers. Then there exists a protocol π' using m clients and n servers that $(t, O(|C|/|\mathbb{F}|))$-securely computes C with abort. Moreover, the communication complexity of π' is bigger than the communication complexity π by a constant factor.*

Proof. First observe that Construction 5.4 can be instantiated (using the AMD code from Theorem 2.2, and the circuit transformation from Theorem 3.7) to produce an $(m, O(|C|/|\mathbb{F}|))$-secure protocol π'' for computing C with abort in the $\widetilde{f}_{\widehat{C}^{\mathsf{AMD}}}$-hybrid model.

Next, notice that by Theorem 5.2 we have that the protocol π when invoked on $\widehat{C}^{\mathsf{AMD}}$ is a t-secure for computing $\widetilde{f}_{\widehat{C}^{\mathsf{AMD}}}$. Thus, obtain π' by replacing the oracle call to $\widetilde{f}_{\widehat{C}^{\mathsf{AMD}}}$ of π'' with the protocol π for $\widehat{C}^{\mathsf{AMD}}$. Since $|\widehat{C}^{\mathsf{AMD}}| = O(|C|)$, we have that the communication complexity of π' is bigger than the communication complexity of π by a constant factor. ■

5.7 The semi-honest BGW protocol

In this section we prove that the semi-honest BGW protocol [BGW88] for computing a circuit C, when executed in the presence of an active adversary, actually computes the additively corruptible version $f_{\widetilde{C}}$ of C (as defined in Definition 5.1). Combining this observation with the results of Section 5.6, we obtain a simple version of the feasibility result of [RB89]. We do this by proving that the semi-honest BGW protocol meets of the requirements of Theorem 5.2.

We begin by describing the BGW semi-honest protocol (see [BGW88, AL11] for details) adapted to the client-server model.

Construction 5.5 (The semi-honest BGW protocol Π^C_{BGW})**.** *Let $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_m} \to \mathbb{F}^{O_1}$ be an m-client circuit and let $n = 2t + 1$ be a positive integer. Denote the gates of C by $g^1, \cdots, g^{|C|}$ and assume that these gates are sorted by topological order. In addition, let $(\mathsf{share}, \mathsf{recover})$ be the Shamir secret sharing scheme from Construction 5.1. Define the following protocol Π^C_{BGW} for C using m clients $C_1, \cdots, C_m$ and n servers $S_1, \cdots, S_n$ as follows.*

1. **Randomness generation phase.** *For each randomness gate g^c the protocol proceeds as follows:*
   (a) *Every server S_i generates a random value $r^{c,i}$ and computes $(m_1^{c,i}, \cdots, m_n^{c,i}) \leftarrow \mathsf{share}(r^{c,i}, t, n)$.*
   (b) *Every server S_i sends to every server S_k (including himself) the message $m_k^{c,i}$.*
   (c) *Every server S_i upon receiving the messages $(m_i^{c,1}, \cdots, m_i^{c,n})$ computes $g_i^c \leftarrow \sum_{k=1}^n m_i^{c,k}$.*
2. **Input sharing phase.** *For each input gate g^c belonging to a client C_i, the client C_i computes $(g_1^c, \cdots, g_n^c) \leftarrow \mathsf{share}(x^c, t, n)$ where x^c is the input value for the c-th input gate and for all $1 \leq k \leq n$ sends g_k^c to S_k.*
3. **Circuit evaluation phase.** *For each gate g^c, $c = 1, \cdots, |C|$, if g^c is a gate with inputs g^a and g^b perform the following:*
   (a) *If g^c is an addition gate, each server S_i computes a share of the output locally as $g_i^c \leftarrow g_i^a + g_i^b$. Similarly, if g^c is a subtraction gate, each server S_i locally computes $g_i^c \leftarrow g_i^a - g_i^b$.*
   (b) *If g^c is a multiplication gate, then each server S_i performs the following on its shares g_i^a and g_i^b:*
      i. *Compute $c_i \leftarrow g_i^a \cdot g_i^b$.*
      ii. *Compute $(c'_{i,1}, \cdots, c'_{i,n}) \leftarrow \mathsf{share}(c_i, t, n)$.*
      iii. *Send to every server S_k (including S_i) the message $m_{i,k} \leftarrow c'_{i,k}$.*
      iv. *Let $(m_{1,i}, \cdots, m_{n,i})$ be the messages received from all the servers by the server S_i. S_i computes $g_i^c \leftarrow \sum_{k=1}^n \Delta_k m_{k,i}$ where $\Delta_1, \cdots, \Delta_n$ are the degree $2t$ Lagrange interpolation coefficients as defined in Definition 5.5.*
4. **Output recovery phase.** *At the end of the computation, for each output gate z each server S_i holds a share g_i^z of the z-th output of $C(\vec{x})$. Servers $S_1, \cdots, S_{t+1}$ send their shares $(g_1^z, \cdots, g_{t+1}^z)$ to client C_1, who then recovers the value of g^z by computing $g^z \leftarrow \mathsf{recover}_{\{S_1, \cdots, S_{t+1}\}}(g_1^z, \cdots, g_{t+1}^z)$.*

The works of [BGW88, AL11] analyzed the semi-honest BGW protocol in the plain model. We now state their result adapted to the client-server model.

Theorem 5.4 ([BGW88, AL11])**.** *For any m-client circuit $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_m} \times \to \mathbb{F}^{O_1}$, the protocol Π^C_{BGW} for computing C using $m = 2t + 1$ servers is private against any passive adversary controlling at most t servers.*

In the following, unless stated otherwise, the term "BGW protocol" will always refer to the protocol in Construction 5.5.

Lemma 5.4 claims that the semi-honest BGW protocol Π^C_{BGW} is linear-based and weakly-private.

Lemma 5.4 (The semi-honest BGW protocol in the malicious model)**.** *Let $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_m} \to \mathbb{F}^{O_1}$ be an m-client circuit and let n, t be positive integers such that $n = 2t+1$. The protocol Π^C_{BGW} defined in Construction 5.5 is linear-based with respect to Shamir secret sharing scheme and weakly-private against active adversaries controlling at most t-servers.*

Proof. We begin by claiming that Π^C_{BGW} is linear-based with respect to Shamir secret sharing scheme. First, notice that Shamir secret sharing scheme is a redundant dense linear secret sharing scheme. Second, notice that Π^C_{BGW} can be viewed as a linear-based protocol with an empty setup phase and where the π_{mult} protocol does not get any auxiliary inputs. Thus Requirement 1 of Definition 5.10 holds vacuously. Finally notice that, Requirements 2, 3, 4a, 4b and 5 of Definition 5.10 can be easily verified from the construction of Π^C_{BGW}.

We now claim that Π^C_{BGW} is weakly-private against active adversaries controlling at most t-servers and an arbitrary number of clients. Indeed, let Adv be an adversary controlling a set of clients $\mathcal{C}$ and a set of servers $\mathcal{S}$ such that $|\mathcal{S}| \leq t$. Notice that the view of the adversary excluding the last communication round during a real execution of the protocol on some input $\vec{x}$ consists of the deterministic inputs $x_{\mathcal{C}}$ of the clients in $\mathcal{C}$, the randomness inputs of the clients in $\mathcal{C}$ and servers in $\mathcal{S}$ as well as the messages obtained by Adv in Steps 1, 2 and 3(b)iii of Construction 5.5.

Notice that in both types of messages, the messages sent by each server or client to all the servers are always a sharing of some value. Next, since $|\mathcal{S}| \leq t$, by the privacy property of Shamir secret sharing with privacy threshold t, the distribution of messages received by the adversary from every honest client or server during every such round of communication does not depend on the value the honest client or server shared. Thus, these messages can be simulated by just sharing some value (say 0) using Shamir secret sharing with privacy threshold t and sending the adversary the resulting shares corresponding to the servers in $\mathcal{S}$. ■

Next, combining the result that the semi-honest BGW protocol Π^C_{BGW} is linear-based and weakly-private (Lemma 5.4) with the result that any linear-based and weakly-private protocol for privately computing a circuit C is a secure protocol for computing the additively corruptible version $\widetilde{f}_C$ (Theorem 5.2), we get the following corollary

Corollary 5.3 (The BGW protocol in the malicious model)**.** *Let n, t be positive integers such that $n = 2t + 1$ and let $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_m} \to \mathbb{F}^{O_1}$ be an m-client circuit. The protocol Π^C_{BGW} defined in Construction 5.5 using m clients and n servers is a t-secure protocol for computing $\widetilde{f}_C$ as defined in Definition 5.1.*

The following corollary states the result of applying our methodology to the semi-honest BGW protocol from Construction 5.5.

Corollary 5.4. *Let n, t be positive integers such that $n = 2t+1$, let $\mathbb{F}$ be a finite field such that $|\mathbb{F}| > n$ and let $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_m} \to \mathbb{F}^{O_1}$ be an m-client circuit. Then there exists an m-client, n-server protocol π that (t, ϵ)-securely computes C with abort for $\epsilon = O(|C|/|\mathbb{F}|)$ where the communication complexity of π is $O(n^2|C|)$ field elements.*

Proof. From Lemma 5.4 it holds that Π^C_{BGW} meets all the requirements of Corollary 5.2. Thus, π can be obtained by instantiating Corollary 5.2 with Π^C_{BGW}. ■

Corollary 5.4 above only guarantees the existence of a protocol π for (t, ϵ)-securely computing a circuit C with abort where $\epsilon = O(|C|/|\mathbb{F}|)$. Moreover, it requires C to only provide outputs for the first

client C_1. Corollary 5.5 below tackles both of these issues. Intuitively, this can be done as follows. First, similarly to the transformation presented in Theorem 3.1, we amplify the security of π by computing C over a suitable extension field. Next, in order to support the case where not only the first client receives output, we use the generic reduction discussed in Section 5.1.

Corollary 5.5. *Let n, t, σ be positive integers such that $n = 2t + 1$, let $\mathbb{F}$ be a finite field and let $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_m} \to \mathbb{F}^{O_1} \times \cdots \times \mathbb{F}^{O_m}$ be an m-client circuit such that $|C| \geq \ell \log |\mathbb{F}|$ where $\ell = \sum_{i=1}^{m} I_i$. Then, there exists an m-client, n-server protocol π that $(t, O(2^{-\sigma} \cdot |C|))$-securely computes C with abort where the communication complexity of π is $O\left(n^2 \cdot \sigma \cdot |C|\right)$ field elements.*

Proof. We prove the special case where only C_1 gets an output from π, that is the circuit C is of the form $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_m} \to \mathbb{F}^{O_1}$. The proof for the general case can be reduced to this special case using the transformation described in Section 5.1.

Let $\mathbb{H}$ be an extension field of $\mathbb{F}$ such that $|\mathbb{H}| \geq 2^{\sigma}$ and $|\mathbb{H}| > n$. Let $B(x) = x(1 - x)$ be the polynomial that vanishes only on 0 and 1, notice that for any $x \in \mathbb{H}$ it holds that $B\left(x^{|\mathbb{F}|-1}\right) = 0$ if and only if $x \in \mathbb{F}$.

Define a circuit $C'(\vec{x}) = (0, C(\vec{x})) + \left(\sum_{i=1}^{n} s_i \cdot B\left(x_i^{|\mathbb{F}|-1}\right)\right) \cdot \vec{r}$ over $\mathbb{H}$ where $s_1, \cdots, s_n$ are random field elements generated using randomness gates inside C' and $\vec{r}$ is a random vector over $\mathbb{H}^{O_1+1}$ generated using random gates inside C'. Notice that for every $\vec{x} \in \mathbb{F}^{\ell}$ it holds that $C'(\vec{x}) = (0, C(\vec{x}))$ and that the output of C' is uniformly random for the case where $\vec{x} \notin \mathbb{F}^{\ell}$. In addition, by construction we have that $|C'| = O(|C| + \ell \log |\mathbb{F}|)$.

Notice that $|\mathbb{H}| > n$ a and that $|C| > \ell \log |\mathbb{F}|$. Thus, by Corollary 5.4 we have that there exists an m-client, n-server protocol π' that $(t, O(|C'|/|\mathbb{H}|))$-securely computes C' with abort where communication complexity of π' is $O(n^2|C'|)$ field elements over $\mathbb{H}$. The required protocol π is obtained by modifying π' and asking the first client, C_1, to abort if the first field element in the output of C' is non-zero. Otherwise, C_1 outputs all but the first field element in the output of C'.

We claim that π is $(t, O(2^{-\sigma} \cdot |C|))$-securely computes C with abort where the communication complexity of π is $O\left(n^2 \cdot \sigma \cdot |C|\right)$ field elements over $\mathbb{F}$. Indeed, notice that $|C'| = O(|C| + \ell \log |\mathbb{F}|)$, $|C| > \ell \log |\mathbb{F}|$ and that $|\mathbb{H}| \geq 2^{\sigma}$. Thus, π indeed $(t, O(2^{-\sigma} \cdot |C'|))$-securely computes C with abort. Next, notice that since $|C'| = O(|C|)$, the communication complexity of π is bigger than the communication complexity of π' by a constant factor. Thus, by our choice of $\mathbb{H}$ we have that the communication complexity of π is $O\left(n^2 \cdot \sigma \cdot |C|\right)$ field elements over $\mathbb{F}$. ■

5.8 The semi-honest Damgård-Nielsen protocol

In this section we prove that a simplified version of the semi-honest DN protocol [DN07] for computing a circuit C, when executed in the presence of an active adversary, actually computes the additively corruptible version $f_{\widetilde{C}}$ of C (as defined in Definition 5.1). Combining this observation with the results of Section 5.6, we obtain an MPC protocol for evaluating a circuit C in the presence of an honest majority that is secure against active adversaries and has a communication complexity of $O(n|C| + n^2)$ field elements. This asymptotically matches the communication complexity of the best known passive-secure MPC protocol of [DN07]. This gives a simpler alternative to the recent protocol of [BFO12] and improves its complexity by eliminating a quadratic overhead for each layer of the circuit, as well as a large polynomial additive factor.

We do this by proving that the simplified version of the semi-honest DN protocol meets of the requirements of Theorem 5.2. We start by presenting a protocol that allows n parties to obtain sharings of degree $2t$ and t of ℓ random field elements with communication complexity of $O(n\ell + n^2)$ field elements (see Section 3.1 in [DN07]). Later, this protocol will be used to reduce the communication complexity of the protocol used to evaluate an entire circuit.

Construction 5.6 (Cf. Section 3.1 in [DN07]). *Let $M \in \mathbb{F}^{(t+1)\times n}$ be a super invertible matrix and let* (share, recover) *be the Shamir secret sharing scheme from Construction 5.1. Consider the following n-party protocol* double-random *where on input ℓ each party P_i performs the following steps $\lceil \frac{\ell}{t+1} \rceil$ times:*

1. *Generate a uniformly random value $s^i \in \mathbb{F}$.*
2. *Compute $(s_1^i, \cdots, s_n^i) \leftarrow \mathsf{share}(s^i, t, n)$.*
3. *Compute $(s_1'^i, \cdots, s_n'^i) \leftarrow \mathsf{share}(s^i, 2t, n)$.*
4. *Send each party P_j the shares $(s_j^i, s_j'^i)$.*
5. *Upon receiving from all the parties the shares $(s_i^1, \cdots, s_i^n)$ and $(s_i'^1, \cdots, s_i'^n)$ the party P_i performs the following:*
   (a) *Compute $(r_i^1, \cdots, r_i^{t+1}) \leftarrow M(s_i^1, \cdots, s_i^n)$.*
   (b) *Compute $(R_i^1, \cdots, R_i^{t+1}) \leftarrow M(s_i'^1, \cdots, s_i'^n)$.*
6. *The output of the i-th party P_i is $(r_i^1, \cdots, r_i^{t+1})$ and $(R_i^1, \cdots, R_i^{t+1})$.*

We define the n-party protocol random*(ℓ) to be the same as* double-random*(ℓ) except that Steps 3 and 5b are not executed.*

We now proceed to describe a simplified version of the semi-honest DN protocol adapted to the client-server model.

Construction 5.7 (The semi-honest DN protocol Π_{DN}^C). *Let $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_m} \to \mathbb{F}^{O_1}$ be an m-client circuit and let n be a positive integer. Denote the gates of C by $g^1, \cdots, g^{|C|}$ and assume that these gates are sorted by topological order. In addition, let* (share, recover) *be the Shamir secret sharing scheme from Construction 5.1. Define the following protocol Π_{DN}^C for C using m clients $C_1, \cdots, C_m$ and n servers $S_1, \cdots, S_n$ as follows.*

1. **Setup phase.** *At this phase the servers would like to compute the randomness needed for evaluation of multiplication gates during the protocol. Let ℓ be the number of multiplication gates inside C. All the servers invoke the* double-random*(ℓ) protocol. At the end of this phase, for every multiplication gate g^c every server S_i holds two shares r_i^c and R_i^c.*
2. **Randomness generation phase.** *At this phase the servers would like to compute the shares corresponding to the outputs of the randomness gates inside C. Let ℓ' be the number of randomness gates inside C. All the servers invoke the* random*(ℓ') protocol. At the end of this phase, for every randomness gate g^c every server S_i holds a share g_i^c corresponding to the output of g^c.*
3. **Input sharing phase.** *For each input gate g^c belonging to a client C_i, the client C_i computes $(g_1^c, \cdots, g_n^c) \leftarrow \mathsf{share}(x^c, t, n)$ where x^c is the input value for the c-th input gate and for all $1 \leq k \leq n$ sends g_k^c to S_k.*
4. **Circuit evaluation phase.** *For each gate g^c, $c = 1, \cdots, |C|$, if g^c is a gate with inputs g^a and g^b perform the following:*
   (a) *If g^c is an addition gate, each server S_i computes a share of the output locally as $g_i^c \leftarrow g_i^a + g_i^b$. Similarly, if g^c is a subtraction gate, each server S_i locally computes $g_i^c \leftarrow g_i^a - g_i^b$.*
   (b) *If g^c is a multiplication gate, then each server S_i performs the following on its shares g_i^a and g_i^b:*
      i. *Compute $c_i \leftarrow g_i^a \cdot g_i^b + R_i^c$ and send c_i to S_1.*

ii. S_1 *upon receiving the shares* $(c_1, \cdots, c_n)$ *from all the servers computes* $D \leftarrow \mathsf{recover}_{\{S_1,\cdots,S_n\}}(c_1, \cdots, c_n)$ *and sends* D *to all the servers.*

iii. Each server S_i *upon receiving a value* D *from* S_1 *computes* $g_i^c \leftarrow D - r_i^c$.

5. **Output recovery phase.** *At the end of the computation, for each output gate* g^c *of* C *each server* S_i *holds a share* g_i^c *of the c-th output of* $C(\vec{x})$. *Servers* $S_1, \cdots, S_{t+1}$ *send their shares* $(g_1^c, \cdots, g_{t+1}^c)$ *to client* C_1, *who then recovers the value of* g^c *by computing* $g^c \leftarrow \mathsf{recover}_{\{S_1,\cdots,S_{t+1}\}}(g_1^c, \cdots, g_{t+1}^c)$.

The work of [DN07] analyzed the semi-honest DN protocol in the plain model. We now state their result adapted to the client-server model.

Theorem 5.5 ([DN07] Sections 3.2, 3.3 and 3.4)**.** *For any m-client circuit* $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_m} \times \rightarrow \mathbb{F}^{O_1}$, *the protocol* Π_{DN}^C *for computing* C *using* $n = 2t + 1$ *servers is private against any passive adversary controlling at most* t *servers. Moreover, the communication complexity of* Π_{DN}^C *is* $O(m|C| + m^2)$ *field elements where* $|C|$ *is the number of gates in* C.

In the following, unless stated otherwise, the term "DN protocol" will always refer to the client server model defined in Construction 5.7.[11]

Lemma 5.5 claims that the semi-honest DN protocol Π_{DN}^C is linear-based and weakly-private.

Lemma 5.5 (The semi-honest DN protocol in the malicious model)**.** *Let* $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_m} \rightarrow \mathbb{F}^{O_1}$ *be an m-client circuit and let* $n = 2t+1$ *be a positive integer. The protocol* Π_{DN}^C *defined in Construction 5.7 is linear-based with respect to Shamir secret sharing scheme and weakly-private against active adversaries controlling at most t-servers.*

Proof. We begin by claiming that Π_{DN}^C is linear-based with respect to Shamir secret sharing scheme. First, notice that Shamir secret sharing scheme is a redundant dense linear secret sharing scheme. Second, notice that, Requirements 1, 2, 3, 4a, 4b and 5 of Definition 5.10 can be easily verified from the construction of Π_{DN}^C.

We now claim that Π_{DN}^C is weakly-private against active adversaries controlling at most t-servers and an arbitrary number of clients. Indeed, let Adv be an adversary controlling a set of clients $\mathcal{C}$ and a set of servers $\mathcal{S}$ such that $|\mathcal{S}| \leq t$. Notice that the view of the adversary excluding the last communication round during a real execution of the protocol on some input $\vec{x}$ consists of the input $x_{\mathcal{C}}$ of the clients in $\mathcal{C}$ as well as the messages obtained by Adv in Steps 1, 2, 3, 4(b)ii of Construction 5.7 for the case where S_1 is corrupted and in Steps 1, 2, 3, 4(b)iii of Construction 5.7 for the case where S_1 is honest.

The messages obtained in Steps 1, 2 and 3 of Construction 5.7 are always a Shamir sharing with privacy threshold t or $2t$ of some value. Next, since $|\mathcal{S}| \leq t$, by the privacy property of Shamir secret sharing with privacy threshold t or $2t$, the distribution of messages received by the adversary from every honest client or server during every such round of communication does not depend on the value the honest client or server shared. Thus, these messages can be simulated by just sharing some value (say 0) using Shamir secret sharing with privacy threshold t or $2t$ as needed and sending the adversary the resulting shares corresponding to the servers in $\mathcal{S}$.

We now show that the messages obtained by Adv in Step 4(b)ii for the case S_1 is corrupted can also be simulated. Indeed, notice that even in the presence of an active adversary, the protocol $\mathsf{double\text{-}random}$ used in Construction 5.7 still produces, as part of its outputs for the honest servers, ℓ random Shamir sharings $(R_1^i, \cdots, R_n^i)$ with privacy threshold $2t$ of random field elements $r^1, \cdots, r^\ell$ where every honest server holds a single share. Moreover, for every i the distributions of both r^i and $(R_1^i, \cdots, R_n^i)$ are independent from all other r_j's and $(R_1^j, \cdots, R_n^j)$ and are also independent from all other messages obtained by the adversary during $\mathsf{double\text{-}random}$. Thus, the messages obtained by S_1 in Step 4(b)ii

[11] Our results also hold in the presence of passive adversaries; thus (to the best of the authors' knowledge) this is the first published complete proof of the semi-honest results of [DN07].

of Construction 5.7 are random Shamir sharings with privacy threshold $2t$ of random field elements. Thus, for the case where S_1 is corrupted, the messages obtained by the adversary in Step 4(b)ii of Construction 5.7 can be easily simulated by generating a random field element r and sending its sharing using Shamir secret sharing with privacy threshold $2t$ to the adversary.

Finally, we now show that the messages obtained by Adv in Step 4(b)iii for the case S_1 is honest can also be simulated. Indeed, notice that the value D computed in Step 4(b)ii of Construction 5.7 is the recovery of the shares obtained by S_1 Step 4(b)ii of Construction 5.7. These shares are constructed in Step 4b by blinding the result of the local multiplication $g_i^a \cdot g_i^b$ by a degree $2t$ Shamir sharing of a random field element. Thus, distribution of D is identical to a uniform distribution over the field and therefore can be simulated by generating a random field element and sending it to the adversary. ■

Next, combining the result that the semi-honest DN protocol Π^C_{DN} is linear-based and weakly-private (Lemma 5.5) with the result that any linear-based and weakly-private protocol for privately computing a circuit C is a secure protocol for computing the additively corruptible version $\widetilde{f}_C$ (Theorem 5.2), we get the following corollary

Corollary 5.6 (The DN protocol in the malicious model)**.** *Let n, t be positive integers such that $n = 2t+1$ and let $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_m} \to \mathbb{F}^{O_1}$ be an m-client circuit. The protocol Π^C_{DN} defined in Construction 5.7 using m clients and n servers is a t-secure protocol for computing $\widetilde{f}_C$ as defined in Definition 5.1.*

The following corollary states the result of applying the our methodology to the semi-honest DN protocol from Construction 5.7. We obtain a protocol π for securely computing a circuit C in the presence of an active adversary for the case of an honest majority. Moreover, π has the same communication complexity as the semi-honest DN protocol of $O(n|C| + n^2)$ field elements where n is the number of servers participating in the protocol.

Corollary 5.7. *Let n, t be positive integers such that $n = 2t + 1$, let $\mathbb{F}$ be a finite field such that $|\mathbb{F}| > n$ and let $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_m} \to \mathbb{F}^{O_1}$ be an m-client circuit. Then there exists an m-client, n-server protocol π that (t, ϵ)-securely computes C with abort for $\epsilon = O(|C|/|\mathbb{F}|)$ where the communication complexity of π is $O(n|C| + n^2)$ field elements.*

Proof. From Lemma 5.5 it holds that Π^C_{DN} meets all the requirements of Corollary 5.2. Thus, π can be obtained by instantiating Corollary 5.2 with Π^C_{DN}. ■

Using the transformations in the proof of Corollary 5.5, Corollary 5.7 can be extended to multi-output functionalities and have its security guarantee be independent of the size of $\mathbb{F}$.

Corollary 5.8. *Let n, t, σ be positive integers such that $n = 2t + 1$, let $\mathbb{F}$ be a finite field and let $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_m} \to \mathbb{F}^{O_1} \times \cdots \times \mathbb{F}^{O_m}$ be an m-client circuit such that $|C| \geq \ell \log |\mathbb{F}|$ where $\ell = \sum_{i=1}^{m} I_i$. Then there exists an m-client, n-server protocol that $(t, O(2^{-\sigma} \cdot |C|))$-securely computes C with abort where the communication complexity of π is $O\left(n\sigma|C| + n^2\sigma\right)$*

5.9 The semi-honest GMW protocol

In this section we tackle the task of secure multiparty computation without honest majority. Since this task is impossible to achieve in the plain model for arbitrary circuits, we are forced to use some kind of hybrid model or have an honestly-executed input-independent preprocessing phase which is done before the execution of the protocol. In this section we demonstrate our methodology to MPC protocols using the former approach while in Section 5.10 we demonstrate the latter. Concretely, Goldriech et al. [GMW87, Gol04] give a simple protocol for boolean circuits in the oblivious transfer (OT) hybrid model. The protocol is secure against a passive adversary who may corrupt an arbitrary subset of the parties. Here we apply a natural extension of this protocol to the arithmetic setting [IPS09], where the OT oracle is replaced by oblivious linear function evaluation (OLE) [NP06].

The OLE functionality. The OLE functionality involves two parties called *sender* and *receiver*. The sender's input consists of a pair of coefficients $a, b \in \mathbb{F}$ representing a linear function $f(x) = ax + b$ over the field $\mathbb{F}$. The receiver's input is a field element $s \in \mathbb{F}$. In response, the functionality outputs $f(s)$ to the receiver and $\perp$ to the sender.

Definition 5.12 (The OLE functionality)**.** *Let $\mathbb{F}$ be a finite field. We define the functionality f_{OLE} that on inputs $(a, b) \in \mathbb{F}^2$ from the sender and $x \in \mathbb{F}$ from the receiver outputs $\perp$ to the sender and $a \cdot x + b$ to the receiver.*

We now proceed to describe an arithmetic version of the GMW protocol in the OLE-hybrid model [GMW87, IPS09].

Construction 5.8 (The arithmetic GMW protocol in the OLE-hybrid model)**.** *Let $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_n} \to \mathbb{F}^{O_1}$ be an n-party circuit. Denote the gates of C by $g^1, \cdots, g^{|C|}$ and assume that these gates are sorted by topological. Define the following protocol Π^C_{GMW} for C in the f_{OLE}-hybrid model as follows.*

1. **Input sharing phase.** *For each input gate g^c belonging to a party P_i, P_i generates $n-1$ random values $g^c_1, \cdots, g^c_{n-1} \in \mathbb{F}$, computes $g^c_n \leftarrow x_c - \sum_{t=1}^{n-1} g^c_t$ where x_c is P_i's input for g^c and sends each party P_k the share g^c_k.*

2. **Randomness generation phase.** *For each random gate g^c every party P_i generates a random share $g^c_i \in \mathbb{F}$.*

3. **Circuit evaluation phase.** *For each gate g^c, $c = 1, \cdots, |C|$, if g^c is a gate with inputs g^a and g^b perform the following:*

   (a) *If g^c is an addition gate, each party P_i computes a share of the output locally as $g^c_i \leftarrow g^a_i + g^b_i$. Similarly, if g^c is a subtraction gate, each party P_i locally computes $g^c_i \leftarrow g^a_i - g^b_i$.*

   (b) *If g^c is a multiplication gate, then the protocol proceeds as follows.*

      i. *Each ordered pair of parties P_i, P_j, such that $i \neq j$ performs the following. First, P_i generates a random value $r_{i,j} \in \mathbb{F}$ and acting as a sender sends $(g^a_i, r_{i,j})$ to the OLE oracle. Next, P_j acting as a receiver sends g^b_j to the OLE oracle. The OLE oracle responds with $s_{i,j} \leftarrow g^a_i \cdot g^b_j + r_{i,j}$ to P_j.*

      ii. *Each party P_i computes $g^c_i \leftarrow g^a_i \cdot g^b_i + \sum_{k \in [n] \setminus \{i\}} (s_{k,i} - r_{i,k})$.*

4. **Output recovery phase.** *At the end of the computation, for each output gate z each party P_i holds a share g^z_i of the z-th output of $C(\vec{x})$. Parties $P_2, \cdots, P_n$ send their shares $g^z_2, \cdots, g^z_n$ to P_1, who then recovers the value of g^z by computing $g^z \leftarrow \sum_{i=1}^{n} g^z_i$.*

Theorem 5.6 ([GMW87, IPS09])**.** *For any n-party circuit $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_n} \times \to \mathbb{F}^{O_1}$, the protocol Π^C_{GMW} defined in Construction 5.8 is private in the presence of any passive adversary.*

We would like to show that when the semi-honest GMW protocol for a circuit C is executed in the presence of an active adversary, it in fact computes the additively corruptible version $\widetilde{f}_C$ of C as defined in Definition 5.1. We start by claiming that Π^C_{GMW} is weakly-private against active adversaries.

Lemma 5.6. *For any n-party circuit C, the protocol Π^C_{GMW} defined in Construction 5.8 is weakly-private against any active adversary.*

Proof. Let Adv be an active adversary controlling a set of parties T. If $|T| = n$ then the lemma is immediate, otherwise we have that $|T| \leq n - 1$. Recall that weak privacy does not consider messages of the last round. Thus, it suffices to show a simulator which generates the messages obtained by the

adversary from honest parties in Step 1 together with the messages obtained by the adversary from the OLE oracle invocations of Step 3 in which an honest party acts as a sender and a corrupted party as a receiver. It is easy to verify that on any input x, these messages are distributed uniformly and independently over $\mathbb{F}$. ■

We now show that the semi-honest GMW protocol for C securely computes the additively corruptible version $\widetilde{f}_C$ of C when executed in the malicious model.

Lemma 5.7 (The GMW protocol in the malicious model)**.** *Let $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_n} \to \mathbb{F}^{O_1}$ be an n-party circuit. The protocol Π^C_{GMW} defined in Construction 5.8 is a secure protocol for computing $\widetilde{f}_C$ as defined in Definition 5.1 in the OLE-hybrid model.*

In order to prove Theorem 5.7 above, for every adversary we need to present a simulator that simulates the view of the corrupted parties together with the output of the honest parties given a trusted party for computing $\widetilde{f}_C$. Before formally describing the full simulator for the general case, we describe the main idea of the simulator for the two party case. Notice that by the weak-privacy requirement, it is possible to simulate the view of any active adversary (excluding the last communication round).

Simulating the output of the honest parties and the view of Adv during the last communication round is more challenging in the case of active adversaries. In order to achieve this, Sim will use Adv in oder to determine an appropriate additive attack A on C. Using A, and using the output of the trusted party, the simulator will simulate the view of the adversary during the last communication round together with the output of the honest parties.

Notice that during the input sharing phase, the simulator has the shares that the adversary has sent to the honest parties. In addition, since the simulator has the input of the corrupted parties, the simulator can compute by itself a set of shares belonging to the corrupted parties that are consistent with its inputs and the shares sent by the adversary to the honest parties. These shares are useful for computing the adversary's internal state during the circuit evaluation phase.

Next, as the simulation progress, for each multiplication gate g^c with inputs g^a and g^b, each corrupted party sends its shares corresponding to g^a and g^b to the OLE oracle (once acting as a sender and once acting as a receiver).

In addition, the simulator will always will try to compute the so called "honest shares" that the adversary *should have* sent (based on it internal state) to the OLE oracle, if it where to start behaving honestly at the current round. Finally, notice that the difference between the actual shares and the honest shares can be used to determine the corresponding additive attack on the inputs of the multiplication gate. Similarly to the analysis in Section 5.5, it is possible to show by induction that thus computed additive attack indeed "explains" all of the corruptions in the output of P_1 in case its honest and view of Adv in final round in the case where P_1 is corrupted.

We now proceed to formally describe a simulator for the n-party GMW protocol. Consider Construction 5.9 below.

Construction 5.9. *Let Adv be an adversary controlling a set of parties T such that $|T| \leq n-1$. Assume without loss of generality that Adv is deterministic. Define the simulator Sim that on inputs $\vec{x}_T$ (of the corrupted parties controlled by Adv), performs the following.*

1. **View generation phase.** *Let $\mathsf{Sim}_{\mathsf{view}}$ be a simulator as guaranteed by the weak-privacy property of Π^C_{GMW}. Invoke $\mathsf{Sim}_{\mathsf{view}}$ on the inputs x_T and obtain a simulated truncated view u' of the adversary (excluding the last communication round for the case where P_1 is corrupted). Retrieve from u the randomness r'_{Adv} that Adv will be using during the execution of Π. Send x_{Adv} and r'_{Adv} to the adversary and initialize two additive attacks A, B on C*

2. **The input sharing phase.**

(a) *For each input gate g^c that is part of the inputs of the honest party P_i, retrieve from u' the values $(g'^c_j)_{P_j \in T}$ representing* Adv*'s shares for g^c and send it to* Adv.

(b) *For each input gate g^c belonging to the corrupted party P_i, the simulator performs the following:*

i. *Receive a messages $(\widetilde{g}'^c_j)_{P_j \in \overline{T}}$ from* Adv *corresponding to the shares of the honest parties in $\overline{T}$.*

ii. *Let x_c be P_i's input to g^c, generate $(g'^c_j)_{P_j \in T}$ uniformly at random such that*

$$\sum_{P_j \in \overline{T}} \widetilde{g}'^c_j + \sum_{P_j \in T} g'^c_j = x_c.$$

Notice that these are one (of many possible) vectors of shares that the adversary should be *using if it were to decide to behave honestly during this phase of the protocol.*

3. **Randomness generation phase.** *For each randomness gate g^c retrieve from u' the values $(g'^c_i)_{P_i \in T}$ representing* Adv*'s shares for g^c if it where to behave honestly during the evaluation of randomness gates.*

4. **Circuit evaluation phase.** *For each gate g^c of C with inputs g^a and g^b, proceed as follows:*

Handling addition and subtraction gates. *In this case no communication takes place. For the corrupted party P_i compute $g'^c_i \leftarrow g'^a_i + g'^b_i$ in the case of addition gates and $g'^c_i \leftarrow g'^a_i - g'^b_i$ in the case of subtraction gates.*

Handling multiplication gates. *If g^c be a multiplication gate, the simulation proceeds as follows.*

(a) *For each honest party P_i and corrupted party P_j retrieve from v'_{Adv} the message $s'_{i,j}$ obtained by P_j while P_i and P_j interacted with the OLE oracle where P_i was acting as a sender and P_j was acting as a receiver.*

(b) *For each honest party P_i and corrupted party P_j obtain from* Adv *the message $m'^b_{i,j}$ sent by P_j to the OLE oracle while P_i and P_j interacted with the OLE oracle where P_i was acting as a sender and P_j was acting as a receiver. Send $(s'_{i,j})_{P_i \in \overline{T}, P_j \in T}$ to* Adv.

(c) *For each honest party P_i and corrupted party P_j obtain from* Adv *the message $(m'^a_{j,i}, r'_{j,i})$ sent by P_j to the OLE oracle while P_i and P_j interacted with the OLE oracle where P_j was acting as a sender and P_i was acting as a receiver.*

(d) *We split the simulation into two cases based on the values of the messages $(m'^b_{i,j}, m'^a_{j,i})_{P_i \in \overline{T}, P_j \in T}$ obtained in Steps 4a and 4b above.*

i. *If for any corrupted party P_j there exists $\widetilde{g}^a_j \in \mathbb{F}$ and $\widetilde{g}'^b_j \in \mathbb{F}$ such that for every honest party P_i it holds that $\widetilde{g}'^a_j = m'^a_{j,i}$ and $\widetilde{g}'^b_j = m'^b_{i,j}$. In this case, the simulation proceeds as follows.*

A. *Update the additive attack A be setting $A_{a,c} \leftarrow \sum_{P_i \in T}(\widetilde{g}'^a_i - g'^a_i)$ and $A_{b,c} \leftarrow \sum_{P_i \in T}(\widetilde{g}'^b_i - g'^b_i)$.*

B. *For every corrupted party P_j compute $g'^c_j \leftarrow \sum_{P_i \in \overline{T}}(s'_{i,j} - r'_{j,i}) + \sum_{P_k \in T} \widetilde{g}'^a_k \cdot \widetilde{g}'^b_j$.*

ii. *Otherwise, we have that there exists a corrupted party P_j such that for every $\widetilde{g}'^a_j \in \mathbb{F}$ and $\widetilde{g}'^b_j \in \mathbb{F}$ there exists an honest party P_i such that $\widetilde{g}'^a_j \neq m'^a_{j,i}$ or $\widetilde{g}'^b_j \neq m'^b_{i,j}$. In this case, the simulation proceeds as follows.*

A. *Generate a random field element r^c and for any gate g^d connected to the output of g^c, updates the additive attack B by setting $B_{c,d} \leftarrow r^c$.*

B. For every corrupted party P_j set $g'^c_j \leftarrow 0$.

5. **Output recovery phase.** *At the end of the of the circuit evaluation phase, for each output gate g^z every corrupted party $P_i \in T$ holds a share $\widetilde{g}^z_i$ of the supposed output. Recall that only P_1 is learn the output. Thus, there are two cases to consider based on whether P_1 is corrupted or not:*

   - P_1 is corrupted. *In this case only the adversary learns the output. The simulation proceeds as follows:*
     (a) The simulator sets to 0 all the coordinates of A and B that were not previously set. In addition, the simulator sets $A_{\mathsf{out}} = B_{\mathsf{out}} = 0$.
     (b) The simulator computes the additive attack A' such that for every wire (g^a, g^b) in C it holds that $A'_{a,b} = A_{a,b} + B_{a,b}$.
     (c) The simulator invokes the trusted party computing $\widetilde{f}_C$ with the inputs of the corrupted parties and with the aforementioned wire corruptions A'. The trusted party responds to the simulator with the outputs y for P_1.
     (d) For each output gate g^z of C the simulator generates the shares $(g'^z_j)_{P_j \in \overline{T}}$ uniformly at random such that $\sum_{P_j \in T} g'^z_j + \sum_{P_j \in \overline{T}} g'^z_j = y_z$ adds them to u' and sends them to Adv.
     (e) The simulator outputs u'.
   - P_1 is honest. *The simulation proceeds as follows:*
     (a) For each output gate g^z of C and for each corrupted party P_j obtain the share $\widetilde{g}'^z_j$ that P_j sends to P_1.
     (b) For each output gate g^z of C, the simulator updates the additive attack A by setting $(A_{\mathsf{out}})_z \leftarrow \sum_{P_j \in T} (\widetilde{g}'^z_j - g'^z_j)$.
     (c) The simulator sets to 0 all the coordinates of A and B that were not previously set. In addition, the simulator sets $B_{\mathsf{out}} = 0$.
     (d) The simulator computes the additive attack A' such that for every wire (g^a, g^b) in C it holds that $A'_{a,b} = A_{a,b} + B_{a,b}$ and $A'_{\mathsf{out}} = A_{\mathsf{out}} + B_{\mathsf{out}}$.
     (e) The simulator invokes the trusted party computing $\widetilde{f}_C$ with the inputs of the corrupted parties and with the wire corruptions A'.
     (f) The simulator outputs u'.

The following Lemma claims that for any adversary Adv and for any input $\vec{x}$, the simulator Sim constructed in Construction 5.9 correctly simulates the view of Adv together with the output of the honest parties during a real execution of the GMW protocol on inputs x in the presence of Adv.

Lemma 5.8. *For any circuit C and for any adversary Adv in the real world controlling a set of parties T such that $|T| < n$ it holds that for any $\vec{x}$*

$$\mathsf{Ideal}_{\widetilde{f}_C, \mathsf{Sim}, T}(\vec{x}) \equiv \mathsf{Real}^{f_{\mathsf{OLE}}}_{\Pi^C_{\mathsf{GMW}}, \mathsf{Adv}, T}(\vec{x})$$

where Sim is the simulator constructed in Construction 5.9.

Proof. Assume without loss of generality that Adv is deterministic and fix an input $\vec{x}$ for all the parties. Notice that Sim invokes $\mathsf{Sim}_{\mathsf{view}}$ in order to generate U'_T. By Definition 5.11 we have that $\mathsf{Sim}_{\mathsf{view}}$ correctly simulates the view of Adv excluding the last communication round. Thus, Sim correctly simulates the view of Adv excluding the last communication round and therefore we have that $U_T \equiv U'_T$.

It remains to prove that Sim correctly simulates the remaining rounds. For any view u from the support of U_T denote by $(L^u_{\overline{T},T}, O^u_{\overline{T}})$ the random variables $(L_{\overline{T},T}, O_{\overline{T}})$ conditioned on $U_T = u$ during

a real execution of the protocol. Similarly, denote by $(L'^{u}_{\overline{T},T}, O'^{u}_{\overline{T}})$ the random variables $(L'_{\overline{T},T}, O'_{\overline{T}})$ conditioned on $U'_T = u$ during an ideal execution of the protocol. Since $\mathsf{Real}_{\Pi,\mathsf{Adv},T}(\vec{x}) = (U_T, L_{\overline{T},T}, O_{\overline{T}})$ and $\mathsf{Ideal}_{\widetilde{f}_C,\mathsf{Sim},T}(\vec{x}) = (U'_T, L'_{\overline{T},T}, O'_{\overline{T}})$ it is therefore sufficient to prove that for any view u it holds that $(L^{u}_{\overline{T},T}, O^{u}_{\overline{T}}) \equiv (L'^{u}_{\overline{T},T}, O'^{u}_{\overline{T}})$.

We now proceed to analyze the input sharing, randomness generation and circuit evaluation phases of Π.

Let $g^1, \cdots, g^{|C|}$ be a topological ordering of the gates of C starting from the input gates and ending with the output gates. For any view u in the support of U'_T and for any gate $g^c \in C$ we denote by $\widetilde{Y}^{c}_{\overline{T},u}$ the shares held by the honest parties corresponding to the output of g^c during a real execution of the protocol conditioned on $U'_{\overline{T}} = u$.

In addition, for any $1 \leq c \leq |C|$, and for any view u in the support of U'_T denote by Z^c_u the random variable corresponding to the value of the output wire of g^c during an ideal execution of the protocol (as computed by $\widetilde{f}_C$) conditioned on $U'_T = u$. Also, for any view u in the support of U'_T denote by B^u the additive attack B computed by the simulator conditioned on $U'_T = u$. By construction, for a gate g^a and for any pair of gates $g^b, g^{b'}$ connected to the output of g^a it holds that $B^u_{a,b} = B^u_{a,b'}$. Thus, we denote by B^a_u the value $B^u_{a,b}$ for every gate g^b connected to the output of g^a.

Moreover, for any view u in the support of U'_T denote by $y^c_{T,u}$ the variables $(g'^c_k)_{P_k \in T}$ generated in Steps 2a, 2(b)i, 3, 4(d)iB, 4(d)iiB and during the processing of addition and subtraction gates of the simulator. Intuitively, these variables represent shares of the output of g^c that the adversary should have if it where to start behaving honestly during the evaluation of the output of g^c. Notice that these shares are completely determined by u and the deterministic adversary Adv.

Finally, denote by $\mathsf{recover}\,(s_1, \cdots, s_n)$ the result of recovering the value from the shares $(s_1, \cdots, s_n)$, namely $\mathsf{recover}\,(s_1, \cdots, s_n) = \sum_{i=1}^{n} s_i$.

The following claim (which can be proved by induction) states that the shares $(\widetilde{Y}^1_{\overline{T},u}, \cdots, \widetilde{Y}^c_{\overline{T},u})$ held by the honest parties after a real execution of the protocol in the presence of Adv as well as the shares $\left(y^1_{T,u}, \cdots, y^{|C|}_{T,u}\right)$ as computed by the simulator correspond to the values $(Z^1_u + B^1_u, \cdots, Z^{|C|}_u + B^{|C|}_u)$ of the internal wires inside $\widetilde{C}$ when attacked by the additive attack A'.

Claim 5.5. *For any view u in the support of U_T, it holds that*

$$\left(Z^1_u + B^1_u, \cdots, Z^{|C|}_u + B^{|C|}_u\right) \equiv \left(\mathsf{recover}((\widetilde{Y}^1_{\overline{T},u}, y^1_{T,u})), \cdots, \mathsf{recover}((\widetilde{Y}^{|C|}_{\overline{T},u}, y^{|C|}_{T,u}))\right).$$

With Claim 5.5 in hand it can be easily verified that for any circuit C and for any adversary Adv in the real world controlling a set of parties T such that $|T| < n$ it holds that for any $\vec{x}$

$$\mathsf{Ideal}_{\widetilde{f}_C,\mathsf{Sim},T}(\vec{x}) \equiv \mathsf{Real}^{f_{\mathsf{OLE}}}_{\Pi^C_{\mathsf{GMW}},\mathsf{Adv},T}(\vec{x}).$$

■

The following theorem states the result of applying the our methodology to the protocol from Construction 5.8. We obtain a protocol allowing any constant number of parties to evaluate an arithmetic circuit in the OLE-hybrid model with security against an active adversary corrupting an arbitrary number of parties. Moreover, π requires only a constant number of OLE calls per gate.

Theorem 5.7. *For any n-party circuit $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_n} \to \mathbb{F}^{O_1}$ there exists a protocol π for ϵ-securely computing C with abort in the OLE hybrid model where $\epsilon = O(|C|/|\mathbb{F}|)$. Moreover π invokes the OLE oracle $O(n^2|C|)$ times and has a total communication complexity of $O(n^2|C|)$ field elements.*

Proof. The proof is similar to the proof of Corollary 5.2. Apply Construction 5.3 on C and obtain the circuit C^{AMD}. Next, let $\widehat{C}^{\mathsf{AMD}}$ be an $O(|C|/|\mathbb{F}|)$-secure implementation of C^{AMD}. Using a variant of Construction 5.4 obtain a $O(|C|/|\mathbb{F}|)$-secure protocol π' for computing C with abort in the $\widetilde{f}_{\widehat{C}^{\mathsf{AMD}}}$-hybrid model. Finally, obtain π by replacing the oracle call of π' with the protocol $\Pi^{\widehat{C}^{\mathsf{AMD}}}_{\mathsf{GMW}}$ from Construction 5.8. ■

Using the transformations in the proof of Corollary 5.5, Theorem 5.7 can be extended to multi-output functionalities and have its security guarantee be independent of the size of $\mathbb{F}$.

Corollary 5.9. *Let n, σ be positive integers, let $\mathbb{F}$ be a finite field and let $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_n} \to \mathbb{F}^{O_1} \times \cdots \times \mathbb{F}^{O_n}$ be n-party circuit such that $|C| \geq \ell \log |\mathbb{F}|$ where $\ell = \sum_{i=1}^{n} I_i$. Then there exists an n-party protocol π that $O\left(2^{-\sigma} \cdot |C|\right)$-securely computes C with abort in the OLE hybrid model where the communication complexity of π is $O\left(n^2 \cdot \sigma \cdot |C|\right)$ field elements.*

5.10 Securing multiparty computation with preprocessing

In this section we present an n-party protocol for securely computing an arithmetic circuit C without honest majority in the plain model. Since this task is impossible for general circuits, our protocol utilizes a preprocessing phase that runs before the computation of C starts and does not depend on the parties inputs to C. Thus, this phase can be modeled as an additional party called the *dealer* that sends each party the corresponding results of the preprocessing phase. Will *will not* require the dealer to behave honestly. Specifically, we allow for two types of adversaries: adversaries that corrupt any number of parties and do not corrupt the dealer or adversaries that only corrupt the dealer and do no corrupt any of the other parties.

Our protocol is constructed as follows. We start from the GMW protocol in the OLE-hybrid model presented in Section 5.9. Next, we replace the OLE oracle used in Construction 5.8 with a random OLE oracle which computes the OLE functionality on random inputs (while making the necessary adaptations to Construction 5.8). Finally, the outputs of the random OLE oracle will be computed by the dealer. Concretely, consider Construction 5.10 below.

Construction 5.10. *Let $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_n} \to \mathbb{F}^{O_1}$ be an n-party circuit containing ℓ multiplication gates. Define the following protocol Π^C for computing C as follows.*

1. **Preprocessing phase.** *At this phase the dealer computes the outputs of the OLE oracle needed the rest of the protocol.*
   (a) *For every multiplication gate g^c and for every pair of parties P_i, P_j such that $P_i \neq P_j$ generate random field elements $m^c_{i,j}, n^c_{i,j}, r^c_{i,j} \in \mathbb{F}$.*
   (b) *For every multiplication gate g^c and for every pair of parties P_i, P_j such that $P_i \neq P_j$ compute $s^c_{i,j} \leftarrow m^c_{i,j} \cdot r^c_{i,j} + n^c_{i,j}$ and send $m^c_{i,j}, n^c_{i,j}$ to P_i and $r^c_{i,j}, s^c_{i,j}$ to P_j.*
2. **Input sharing phase.** *For each input gate g^c belonging to a party P_i, P_i generates $n-1$ random values $g^c_1, \cdots, g^c_{n-1} \in \mathbb{F}$, computes $g^c_n \leftarrow x_c - \sum_{t=1}^{n-1} g^c_t$ and sends each party P_k the share g^c_k where x_c is P_i's input for g^c.*
3. **Randomness generation phase.** *For each random gate g^c every party P_i generates a random share $g^c_i \in \mathbb{F}$.*
4. **Circuit evaluation phase.** *For each gate g^c, $c = 1, \cdots, |C|$, if g^c is a gate with inputs g^a and g^b perform the following:*
   (a) *If g^c is an addition gate, each party P_j computes a share of the output locally as $g^c_j \leftarrow g^a_j + g^b_j$. Similarly, if g^c is a subtraction gate, each party P_j locally computes $g^c_j \leftarrow g^a_j - g^b_j$.*

(b) *If g^c is a multiplication gate, then the protocol proceeds as follows.*

i. *Each ordered pair of parties P_i, P_j such that $P_i \neq P_j$ performs the following:*

A. *P_j computes $u^c_{i,j} \leftarrow g^b_j - r^c_{i,j}$ and sends it to P_i.*

B. *P_i computes $v^c_{i,j} \leftarrow u^c_{i,j} \cdot m^c_{i,j}$, $w^c_{i,j} \leftarrow g^a_i - m^c_{i,j}$ and sends $v^c_{i,j}, w^c_{i,j}$ to P_j. Notice that $v^c_{i,j} = (g^b_j - r^c_{i,j}) \cdot m^c_{i,j}$.*

C. *P_j computes $t^c_{i,j} \leftarrow w^c_{i,j} \cdot g^b_j + v^c_{i,j} + s^c_{i,j}$. Notice that $t^c_{i,j} = g^a_i g^b_j + n^c_{i,j}$.*

ii. *Each party P_i computes $g^c_i \leftarrow g^a_i \cdot g^b_i + \sum_{j \in [n] \setminus \{i\}} (t^c_{j,i} - n^c_{i,j})$.*

5. **Output recovery phase.** *At the end of the computation, for each output gate z each party P_j holds a share g^z_j of the zth output of $C(\vec{x})$. Parties $P_2, \cdots, P_n$ send their shares $g^z_2, \cdots, g^z_n$ to P_1, who then recovers the value of g^z by computing $g^z \leftarrow \sum_{i=1}^n g^z_i$.*

The following theorem can be easily verified

Theorem 5.8. *For any n-party circuit $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_n} \times \rightarrow \mathbb{F}^{O_1}$, the protocol Π^C for computing C defined in Construction 5.10 is private in the presence of any passive adversary controlling any subset of parties or the dealer.*

We start claiming that Π^C is weakly-private against active adversaries. The following theorem can be easily verified.

Lemma 5.9. *For any n-party circuit C, the protocol Π^C for computing C defined in Construction 5.10 is weakly-private against any active adversary controlling the dealer or any subset of parties.*

We now claim that when Π^C securely computes $\widetilde{f}_C$ when invoked against an active adversary controlling (only) the dealer.

Lemma 5.10. *Let $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_n} \rightarrow \mathbb{F}^{O_1}$ be a n-party circuit. The protocol Π^C defined in Construction 5.10 is a secure protocol against any active adversary controlling only the dealer for computing $\widetilde{f}_C$ as defined in Definition 5.1.*

Proof. Let Adv be an adversary controlling the dealer. Assume without loss of generality that Adv is deterministic. Define the simulator Sim that on inputs $\perp$ (of the dealer), performs the following.

1. **View generation phase.** Let $\mathsf{Sim}_{\mathsf{view}}$ be a simulator as guaranteed by the weak-privacy property of Π^C. Invoke $\mathsf{Sim}_{\mathsf{view}}$ on the inputs $\perp$ and obtain a simulated view v'_{Adv} of the adversary.

2. **Circuit evaluation phase.** The simulation proceeds as follows.

   (a) Send v'_{Adv} to Adv.

   (b) For every multiplication gate g^c and for every pair of parties P_i and P_j such that $P_i \neq P_j$ obtain the from Adv the messages $m'^c_{i,j}$ and $n'^c_{i,j}$ sent to P_i in Step 1b of Construction 5.10. In addition, obtain from Adv the messages $r'^c_{i,j}$ and $s'^c_{i,j}$ sent to P_j in Step 1b of Construction 5.10.

   (c) For every multiplication gate g^c and for every pair of parties P_i and P_j such that $P_i \neq P_j$ compute $\delta'^c_{i,j} \leftarrow s'^c_{i,j} - \left(m'^c_{i,j} \cdot r'^c_{i,j} + n'^c_{i,j}\right)$.

   (d) For every multiplication gate g^c compute $\delta^c \leftarrow \sum_{i \neq j} \delta^c_{i,j}$.

   (e) For every multiplication gate g^c and for every gate g^d connected to the output of g^c set $A_{c,d} \leftarrow \delta^c$.

3. **Output recovery phase.** The simulation proceeds as follows.

(a) The simulator sets to 0 all the coordinates of A that were not previously set.

(b) The simulator invokes the trusted party computing $\widetilde{f}_C$ with the aforementioned wire corruptions A.

(c) The simulator outputs v'_{Adv}.

It can easily be verified that for any circuit C, and for any adversary Adv in the real world controlling the dealer it holds that for any $\vec{x}$

$$\mathsf{Ideal}_{\widetilde{f}_C,\mathsf{Sim},\mathrm{dealer}}(\vec{x}) \equiv \mathsf{Real}_{\Pi^C,\mathsf{Adv},\mathrm{dealer}}(\vec{x}).$$

■

We now claim that when Π^C securely computes $\widetilde{f}_C$ when invoked against an active adversary controlling any subset of parties and not the dealer.

Lemma 5.11. *Let $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_n} \to \mathbb{F}^{O_1}$ be a n-party circuit. The protocol Π^C defined in Construction 5.10 is a secure protocol for computing $\widetilde{f}_C$ as defined in Definition 5.1 against any active adversary controlling any subset of parties and not the dealer.*

Proof. Let Adv be an adversary controlling a set of parties T and not the dealer. If $|T| = n$ then the theorem follows immediately. Otherwise, we have that $|T| \leq n-1$. Assume without loss of generality that Adv is deterministic. Define the simulator Sim that on inputs $\vec{x}_T$ (of the corrupted parties controlled by Adv), performs the following.

1. **View generation phase.** Let $\mathsf{Sim}_{\mathsf{view}}$ be a simulator as guaranteed by the weak-privacy property of Π^C. Invoke $\mathsf{Sim}_{\mathsf{view}}$ on the inputs x_{Adv} and obtain a simulated truncated view u'_{Adv} of the adversary (excluding the last communication round for the case where P_1 is corrupted). Retrieve from u'_{Adv} the randomness r'_{Adv} that Adv will be using during the execution of Π. Send x_{Adv} and r'_{Adv} to the adversary and initialize two additive attacks A, B on C.

2. **Preprocessing phase.** For every multiplication gate g^c for every honest party P_i and corrupted party P_j retrieve from u'_{Adv} the values $m'^c_{j,i}, n'^c_{j,i}, r'^c_{i,j}$ and $s'^c_{i,j}$ obtained by the adversary from the dealer during the preprocessing phase and send them to Adv.

3. **The input sharing phase.**

   (a) For each input gate g^c that is part of the inputs of the honest party P_i, retrieve from u'_{Adv} the values $(g'^c_j)_{P_j \in T}$ representing Adv's shares for g^c and send it to Adv.

   (b) For each input gate g^c belonging to the corrupted party P_i, the simulator performs the following:

      i. Receive a messages $(\widetilde{g}'^c_j)_{P_j \in \overline{T}}$ from Adv corresponding to the shares of the honest parties in $\overline{T}$.

      ii. Let x_c be P_i's input to g^c, generate $(g'^c_j)_{P_j \in T}$ uniformly at random such that $\sum_{P_j \in \overline{T}} \widetilde{g}'^c_j + \sum_{P_j \in T} g'^c_j = x_c$. Notice that these are one (of many possible) vectors of shares that the adversary *should be* using if it were to decide to behave honestly during this phase of the protocol.

4. **Randomness generation phase.** For each random gate g^c and for each corrupted party P_i retrieve from u'_{Adv} the values $(g'^c_i)_{P_i \in T}$ representing Adv's shares for g^c.

5. **Circuit evaluation phase.** For each gate g^c of C with inputs g^a and g^b, proceed as follows:

   Handling addition and subtraction gates. In this case no communication takes place. For the corrupted party P_i compute $g_i'^c \leftarrow g_i'^a + g_i'^b$ in the case of addition gates and $g_i'^c \leftarrow g_i'^a - g_i'^b$ in the case of subtraction gates.

   Handling multiplication gates. If g^c be a multiplication gate, the simulation proceeds as follows.

   (a) For every honest party P_i and corrupted party P_j retrieve from u'_{Adv} the value $u_{j,i}'^c$ that P_j receives during Step 4(b)iA of Construction 5.10 and send it to Adv.

   (b) For every honest party P_i and corrupted party P_j obtain from Adv the value $\widetilde{u}_{i,j}'^c$ that P_i receives during Step 4(b)iA of Construction 5.10.

   (c) For every honest party P_i and corrupted party P_j retrieve from u'_{Adv} the values $v_{i,j}'^c, w_{i,j}'^c$ that P_j receives during Step 4(b)iB of Construction 4(b)iB and send it to Adv.

   (d) For every honest party P_i and corrupted party P_j obtain from Adv the values $\widetilde{v}_{j,i}'^c, \widetilde{w}_{j,i}'^c$ that P_i receives during Step 4(b)iB of Construction 5.10.

   (e) We split the simulation into two cases based on the values of $\widetilde{u}_{i,j}'^c$, and $\widetilde{w}_{j,i}'^c$ obtained by the simulator from the adversary in Steps 5b and 5d above.

      i. If for every corrupted party P_j there exists $\widetilde{g}_j'^a, \widetilde{g}_j'^b$ such that for every honest party P_i it holds that $\widetilde{u}_{i,j}'^c = \widetilde{g}_j'^b - r_{i,j}'^c$ and $\widetilde{w}_{j,i}'^c \leftarrow \widetilde{g}_j'^a - m_{j,i}'^c$. In this case the simulation proceeds as follows.

         A. For every corrupted party P_j and honest party P_i compute $v_{j,i}'^c \leftarrow (\widetilde{g}_j'^b - r_{j,i}'^c) \cdot m_{j,i}'^c$.

         B. For every corrupted party P_j and honest party P_i compute $t_{i,j}'^c \leftarrow w_{i,j}'^c \cdot \widetilde{g}_j'^b + v_{i,j}'^c + s_{i,j}'^c$.

         C. Update the additive attack A be computing $A_{a,c} \leftarrow \sum_{P_j \in T}(\widetilde{g}_j'^a - g_j'^a)$ and $A_{b,c} \leftarrow \sum_{P_j \in T}(\widetilde{g}_j'^b - g_j'^b)$.

         D. For any gate g^d connected to the output of g^c update the additive attack B by computing $B_{c,d} \leftarrow \sum_{\substack{P_j \in T \\ P_i \in \overline{T}}} (\widetilde{v}_{j,i}'^c - v_{j,i}'^c)$.

         E. For every corrupted party P_j compute $g_j'^c \leftarrow \sum_{P_i \in \overline{T}}(t_{i,j}'^c - v_{j,i}'^c) + \widetilde{g}_j'^a \widetilde{g}_j'^b$.

      ii. Otherwise, there exists a corrupted party P_j such that for any $\widetilde{g}_j'^a, \widetilde{g}_j'^b$ there exists an honest party P_i such that $\widetilde{u}_{i,j}'^c \neq \widetilde{g}_j'^b - r_{i,j}'^c$ or $\widetilde{w}_{j,i}'^c \neq \widetilde{g}_j'^a - m_{j,i}'^c$. In this case the simulation proceeds as follows.

         A. Generate a random field element r^c and for any gate g^d connected to the output of g^c, updates the additive attack B by setting $B_{c,d} \leftarrow r^c$.

         B. For every corrupted party P_j set $g_j'^c \leftarrow 0$.

6. **Output recovery phase.** At the end of the of the circuit evaluation phase, for each output gate g^z every corrupted party $P_i \in T$ holds a share $\widetilde{g}_i^z$ of the supposed output. Recall that only P_1 is learn the output. Thus, there are two cases to consider based on whether P_1 is corrupted or not:

   - P_1 *is corrupted.* In this case only the adversary learns the output. The simulation proceeds as follows:

     (a) The simulator sets to 0 all the coordinates of A and B that were not previously set. In addition, the simulator sets $A_{\mathsf{out}} = B_{\mathsf{out}} = 0$.

     (b) The simulator computes the additive attack A' such that for every wire (g^a, g^b) in C it holds that $A'_{a,b} = A_{a,b} + B_{a,b}$.

(c) The simulator invokes the trusted party computing $\widetilde{f}_C$ with the inputs of the corrupted parties and with the aforementioned wire corruptions A'. The trusted party responds to the simulator with the outputs y for P_1.

(d) For each output gate g^z of C the simulator generates shares $(g'^z_j)_{P_j \in \overline{T}}$ uniformly at random such that $\sum_{P_j \in T} g'^i_j + \sum_{P_j \in \overline{T}} g'^i_j = y_z$ adds them to u'_{Adv} and sends them to Adv.

(e) The simulator outputs u'_{Adv}.

- P_1 *is honest.* The simulation proceeds as follows:

(a) For each output gate g^z of C and for each corrupted party P_j obtain the share $\widetilde{g}'^z_j$ that P_j sends to P_1.

(b) For each output gate g^z of C, the simulator updates the additive attack A by setting $(A_{\mathsf{out}})_z \leftarrow \sum_{P_j \in T} (\widetilde{g}'^z_j - g'^z_j)$.

(c) The simulator sets to 0 all the coordinates of A and B that were not previously set. In addition, the simulator sets $B_{\mathsf{out}} = 0$.

(d) The simulator computes the additive attack A' such that for every wire (g^a, g^b) in C it holds that $A'_{a,b} = A_{a,b} + B_{a,b}$ and $A'_{\mathsf{out}} = A_{\mathsf{out}} + B_{\mathsf{out}}$.

(e) The simulator invokes the trusted party computing $\widetilde{f}_C$ with the inputs of the corrupted parties and with the wire corruptions A' computed so far.

(f) The simulator outputs u'_{Adv}.

It can be verified that for any circuit C, and for any adversary Adv in the real world controlling a set of parties T and not the dealer such that $|T| \leq n-1$ it holds that for any $\vec{x}$

$$\mathsf{Ideal}_{\widetilde{f}_C, \mathsf{Sim}, T}(\vec{x}) \equiv \mathsf{Real}_{\Pi^C, \mathsf{Adv}, T}(\vec{x}).$$

■

We now state the following corollary the proof of which immediately follows from Lemmas 5.10 and 5.11.

Corollary 5.10. *Let $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_n} \to \mathbb{F}^{O_1}$ be a n-party circuit. The protocol Π^C defined in Construction 5.10 is a secure protocol for computing $\widetilde{f}_C$ as defined in Definition 5.1 in the presence of any active adversary controlling the dealer or any subset of parties.*

The following theorem states the result of applying our methodology to the protocol from Construction 5.10. We obtain a protocol that is secure in the presence of an active adversary controlling any subset of parties or the dealer.

Theorem 5.9. *For any n-party circuit $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_n} \to \mathbb{F}^{O_1}$ there exists a protocol π using n parties and an additional special party called the dealer such that is a $O(|C|/|\mathbb{F}|)$-secure protocol for computing C with abort in the presence of any active adversary controlling the dealer or any subset of parties. Moreover, the communication complexity of π is $O(n^2|C|)$*

Proof. Apply Construction 5.3 on C and obtain the circuit C^{AMD}. In addition, let $\widehat{C}^{\mathsf{AMD}}$ be an $O(|C|/|\mathbb{F}|)$-secure implementation of C^{AMD}. Using a variant of Construction 5.4 obtain a $O(|C|/|\mathbb{F}|)$-secure protocol π' for computing C with abort in the presence of any active adversary controlling the dealer or any subset of parties in the $\widetilde{f}_{\widehat{C}^{\mathsf{AMD}}}$-hybrid model. Finally, obtain π by replacing the oracle call of π' with the protocol $\Pi^{\widehat{C}^{\mathsf{AMD}}}$ from Construction 5.10. ■

Using the transformations in the proof of Corollary 5.5, Theorem 5.9 can be extended to multi-output functionalities and have its security guarantee be independent of the size of $\mathbb{F}$.

Corollary 5.11. *Let n, σ be a positive integer, let $\mathbb{F}$ be a finite field and let $C : \mathbb{F}^{I_1} \times \cdots \times \mathbb{F}^{I_n} \to \mathbb{F}^{O_1} \times \cdots \times \mathbb{F}^{O_n}$ be an n-party circuit such that $|C| \geq \ell \log |\mathbb{F}|$ where $\ell = \sum_{i=1}^{n} I_i$. Then there exists a protocol π using n parties and and an additional special party called the dealer that $O(2^{-\sigma} \cdot |C|)$-securely computes C with abort in the presence of any active adversary controlling the dealer or any subset of the parties. Moreover the communication complexity of π is $O\left(n^2 \cdot \sigma \cdot |C|\right)$ field elements.*

Appendices

A A MIP-based construction

In this section, we present a construction for additive correctness which is based on multiprover interactive proofs (MIP) systems. While the main advantage of this construction is its high efficiency for circuits with short succinct descriptions, the construction will not achieve full security but only correctness with a decoder. Moreover, since the size of the resulting decoder will be linear in the size of the circuit description, the construction is only meaningful for circuits with small descriptions. Finally, the size of the resulting decoder is also linear in the input size of the original circuit. This limitation prevents improving this construction as done in Section 4.2.2 by applying it to every layer of the original circuit, treating the obtained decoder as part of the next layer.

Despite the above limitations, for a circuit $C : \{0,1\}^n \to \{0,1\}^k$ with a succinct description F (see Definition A.1), we present a construction which produces AMD circuits of size $|\widehat{C}| = (\sigma + |C| \cdot |F|) \cdot \text{polylog}(\sigma, |C|, |F|)$ and $|D| = (n + k + |F|) \cdot \sigma \cdot \text{polylog}(|C|, |F|)$ where σ is the security parameter. This is compared with Theorem 1.2 where $|\widehat{C}| = |C| \cdot poly(\sigma)$ and $|D| = k \cdot poly(\sigma)$.

We begin by defining the notion of a succinct circuit descriptor.

Definition A.1 (Circuit descriptor, Cf. Definition 2.1 in [BSCGT13])**.** *Let C be a circuit containing 2^n gates with each gate labeled by some n-bit string. In addition let $F : \{0,1\}^{n+2} \to \{0,1\}^n$ be a circuit. We say that F is the* succinct circuit descriptor *of C is the following holds. For any $s \in \{0,1\}^n$ let $s_{00} = F(s00), s_{01} = F(s01), s_{10} = F(s10)$, and $s_{11} = F(s11)$. We require that gates s_{00} and s_{01} have wires going into gate s, and gate s has wires going into s_{10} and s_{11}. If $s = s_{00} = s_{01}$ then s is an input gate and if $s = s_{10} = s_{11}$ then s is an output gate. Notice that $|F| \leq O(|C|)$.*

Basic MIP-based construction. The main idea behind the construction is as follows: we start from a PCP with a polynomial time prover. Next, using the transformation of [TS96] we obtain a constant soundness MIP system from the PCP by putting two non-communicating PCP proves side by side and using one to verify answers of the other. Next, we observe that in order to amplify the soundness of the MIP system, all we need is a circuit that will select additional bits from the PCP proof in response to additional queries from the verifier. The correctness of this construction follows from the fact that even though the adversary can additively corrupt each prover of the MIP system, it cannot cause the provers to communicate. Thus, the soundness of the MIP system is maintained. In fact, notice that this construction is much more robust. As long as the adversary does not corrupt the randomness gate (for example by setting their output to be identically zero) and does not cause the provers to communicate (by moving wires from place to place inside the circuit), the MIP system remains sound and the construction withstands the attack.

Notice that unlike the constructions presented in Section 4.1, we cannot make the guarantee that the soundness of the PCP will be preserved even when the *verifier* is additively attacked. Thus, since the verifier needs to be protected inside the decoder, and since it still needs to store a description of C this construction achieves non trivial parameters only for sufficiently uniform circuits.

Instantiating the PCP to MIP transformation with PCPs with polylogarithmic overhead [BSGH$^+$05, BSS08, BSCGT13] we obtain the following theorem.

Theorem A.1. *For any positive integer σ and for any circuit descriptor F describing a circuit $C : \{0,1\}^n \to \{0,1\}^k$ there exists a $2^{-\sigma}$-correct implementation $(\widehat{C}, D)$ of C such that $|\widehat{C}| = \sigma \cdot |C| \cdot |F| \cdot \mathit{polylog}(|C|, |F|)$ and $|D| = (n + k + |F|) \cdot \sigma \cdot \mathit{polylog}(|C|, |F|)$.*

Efficient MIP-based construction. We now present a more efficient version of the above construction. The construction in Theorem A.1 achieved $2^{-\sigma}$-correctness by running the PCP prover $O(\sigma)$ times each time selecting the necessary bits for the verifier to read. We now improve this part in the construction by utilizing a sorting network which does not require the PCP prover to be invoked $O(\sigma)$ times but only a constant number of times. A sorting network is a directed acyclic graph with disjoint set of *sources* and *sinks*. Each node in the graph is either a source, a sink or a *comparator* node. The input to the sorting network is a list of records where the i-th source node is assigned the i-th record from the list. Each comparator node is connected by inbound edges to two other comparator or source nodes and is connected by an outbound edge to two other comparator or sink nodes. Every comparator node maps its inputs to its outputs using some comparison logic. The guarantee of the entire sorting network is that the sink nodes will contain the sorted list of record that was given to the network as input. The i-th sink node will contain the i-th recored of the result. For a comprehensive reference on sorting networks see Section 3.5 in [Lei92]. *Optimal* sorting networks where presented in the work of Ajtai et al. [AKS83] where a graph of size $O(n \log n)$ is needed to sort n records.

Using sorting networks it is possible to construct a circuit $\mathsf{sel}_{n,k}$ that is able to *efficiently* select k records out of a list of length n such that $|\mathsf{sel}_{n,k}| = O((n + k) \cdot \log(n + k))$. This is compared to the naive solution for this problem using k n-to-1 multiplexers resulting in a circuit of size $O(k \cdot n)$.

Notice that the MIP provers described above simulate the PCP prover P' and select specific coordinates from its output based on the queries given to them as input. Thus, amplifying the soundness of the MIP system used in Theorem A.1 by parallel repetitions where the selection process is implemented using an efficient n-out-of-k selector $\mathsf{sel}_{n,k}$ yields the following theorem.

Theorem A.2. *For any positive integer σ and for any circuit descriptor F describing a circuit $C : \{0,1\}^n \to \{0,1\}^k$ there exists a $2^{-\sigma}$-correct implementation $(\widehat{C}, D)$ of C such that $|\widehat{C}| = (\sigma + |C| \cdot |F|) \cdot \mathit{polylog}(\sigma, |C|, |F|)$ and $|D| = (n + k + |F|) \cdot \sigma \cdot \mathit{polylog}(|C|, |F|)$.*

Acknowledgments

D. Genkin and Y. Ishai were supported by European Union's Tenth Framework Programme (FP10/2010-2016) under grant agreement no. 259426 ERC-CaC. Y. Ishai was additionally supported by ISF grant 1361/10 and BSF grant 2012378. M. Prabhakaran was supported by NSF grants 07-47027 and 12-28856. A. Sahai was supported in part from a DARPA/ONR PROCEED award, NSF grants 1228984, 1136174, 1118096, and 1065276, a Xerox Faculty Research Award, a Google Faculty Research Award, an Intel equipment grant, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014-11-1-0389. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government. E. Tromer was supported, and D. Genkin was additionally supported, by the Check Point Institute for Information Security, the Israeli Centers of Research Excellence I-CORE program (center 4/11), the Israeli Ministry of Science and Technology, and NATO's Public Diplomacy Division in the Framework of "Science for Peace".

References

[AKS83] Miklós Ajtai, János Komlós, and Endre Szemerédi, *An O(n log n) sorting network*, STOC, 1983, pp. 1–9.

[AL11] Gilad Asharov and Yehuda Lindell, *A full proof of the BGW protocol for perfectly-secure multiparty computation*, IACR Cryptology ePrint Archive **2011** (2011), 136.

[ALM+92] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy, *Proof verification and hardness of approximation problems*, FOCS, 1992, pp. 14–23.

[BCG+06] Michael Ben-Or, Claude Crépeau, Daniel Gottesman, Avinatan Hassidim, and Adam Smith, *Secure multiparty quantum computation with (only) a strict honest majority*, FOCS, 2006, pp. 249–260.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias, *Semi-homomorphic encryption and multiparty computation*, EUROCRYPT, 2011, pp. 169–188.

[BFO12] Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky, *Near-linear unconditionally-secure multiparty computation with a dishonest minority*, CRYPTO, 2012, pp. 663–680.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson, *Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract)*, STOC, 1988, pp. 1–10.

[BSCGT13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer, *On the concrete efficiency of probabilistically-checkable proofs*, STOC, 2013, pp. 585–594.

[BSGH+05] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan, *Short pcps verifiable in polylogarithmic time*, IEEE Conference on Computational Complexity, 2005, pp. 120–134.

[BSS08] Eli Ben-Sasson and Madhu Sudan, *Short PCPs with polylog query complexity*, SIAM J. Comput. **38** (2008), no. 2, 551–607.

[Can00] Ran Canetti, *Security and composition of multiparty cryptographic protocols*, J. Cryptology **13** (2000), no. 1, 143–202.

[Can01] ———, *Universally composable security: A new paradigm for cryptographic protocols*, FOCS, 2001, pp. 136–145.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård, *Multiparty unconditionally secure protocols (extended abstract)*, STOC, 1988, pp. 11–19.

[CDF+08] Ronald Cramer, Yevgeniy Dodis, Serge Fehr, Carles Padró, and Daniel Wichs, *Detection of algebraic manipulation with applications to robust secret sharing and fuzzy extractors*, EUROCRYPT, 2008, pp. 471–488.

[CDI05] Ronald Cramer, Ivan Damgård, and Yuval Ishai, *Share conversion, pseudorandom secret-sharing and applications to secure computation*, TCC, 2005, pp. 342–362.

[CW79] Larry Carter and Mark N. Wegman, *Universal classes of hash functions*, J. Comput. Syst. Sci. **18** (1979), no. 2, 143–154.

[DI05] Ivan Damgård and Yuval Ishai, *Constant-round multiparty computation using a black-box pseudorandom generator*, CRYPTO, 2005, pp. 378–394.

[DI06] ———, *Scalable secure multiparty computation*, CRYPTO, 2006, pp. 501–520.

[DIK10] Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard, *Perfectly secure multiparty computation and the computational overhead of cryptography*, EUROCRYPT, 2010, pp. 445–465.

[DK12] Dana Dachman-Soled and Yael Tauman Kalai, *Securing circuits against constant-rate tampering*, CRYPTO, 2012, pp. 533–551.

[DKRS06] Yevgeniy Dodis, Jonathan Katz, Leonid Reyzin, and Adam Smith, *Robust fuzzy extractors and authenticated key agreement from close secrets*, CRYPTO, 2006, pp. 232–250.

[DN07] Ivan Damgård and Jesper Buus Nielsen, *Scalable and unconditionally secure multiparty computation*, CRYPTO, 2007, pp. 572–590.

[DO77] R. Dobrushin and E. Ortyukov, *Upper bound on the redundancy of self-correcting arrangements of unreliable functional elements*, Problems of Information Transmission **23** (1977), no. 2, 203–218.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias, *Multiparty computation from somewhat homomorphic encryption*, CRYPTO, 2012, pp. 643–662.

[DSK14] Dana Dachman-Soled and Yael Tauman Kalai, *Securing circuits and protocols against 1/poly(k) tampering rate*, TCC, 2014, pp. 540–565.

[FPV11] Sebastian Faust, Krzysztof Pietrzak, and Daniele Venturi, *Tamper-proof circuits: How to trade leakage for tamper-resilience*, ICALP, 2011, pp. 391–402.

[Fre77] Rusins Freivalds, *Probabilistic machines can use less running time*, IFIP Congress, 1977, pp. 839–842.

[GIKR02] Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin, *On 2-round secure multiparty computation*, CRYPTO, 2002, pp. 178–193.

[GLM+04] Rosario Gennaro, Anna Lysyanskaya, Tal Malkin, Silvio Micali, and Tal Rabin, *Algorithmic tamper-proof (ATP) security: Theoretical foundations for security against hardware tampering*, TCC, 2004, pp. 258–277.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson, *How to play any mental game or a completeness theorem for protocols with honest majority*, STOC, 1987, pp. 218–229.

[Gol04] Oded Goldreich, *The foundations of cryptography - volume 2, basic applications*, Cambridge University Press, 2004.

[GS95] Anna Gál and Mario Szegedy, *Fault tolerant circuits and probabilistically checkable proofs*, Structure in Complexity Theory Conference, 1995, pp. 65–73.

[HL10] Carmit Hazay and Yehuda Lindell, *Efficient secure two-party protocols - techniques and constructions*, Information Security and Cryptography, Springer, 2010.

[IKHC14] Dai Ikarashi, Ryo Kikuchi, Koki Hamada, and Koji Chida, *Actively private and correct mpc scheme in t<n/2 from passively secure schemes with small overhead*, IACR Cryptology ePrint Archive **2014** (2014), 304.

[IKM+13] Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, Claudio Orlandi, and Anat Paskin-Cherniavsky, *On the power of correlated randomness in secure computation*, TCC, 2013, pp. 600–620.

[IKO+11] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, and Amit Sahai, *Efficient non-interactive secure computation*, EUROCRYPT, 2011, pp. 406–425.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai, *Founding cryptography on oblivious transfer – efficiently*, CRYPTO, 2008, pp. 572–591.

[IPS09] ———, *Secure arithmetic computation with no honest majority*, TCC, 2009, pp. 294–314.

[IPSW06] Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David Wagner, *Private circuits ii: Keeping secrets in tamperable circuits*, EUROCRYPT, 2006, pp. 308–327.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner, *Private circuits: Securing hardware against probing attacks*, CRYPTO, 2003, pp. 463–481.

[Kil88] Joe Kilian, *Founding cryptography on oblivious transfer*, STOC, 1988, pp. 20–31.

[KKS11] Yael Tauman Kalai, Bhavana Kanukurthi, and Amit Sahai, *Cryptography with tamperable and leaky memory*, CRYPTO, 2011, pp. 373–390.

[KLM94] Daniel J. Kleitman, Frank Thomson Leighton, and Yuan Ma, *On the design of reliable boolean circuits that contain partially unreliable gates*, FOCS, 1994, pp. 332–346.

[KLR12] Yael Tauman Kalai, Allison B. Lewko, and Anup Rao, *Formulas resilient to short-circuit errors*, FOCS, 2012, pp. 490–499.

[KN89] Mark G. Karpovsky and Prawat Nagvajara, *Optimal codes for minimax criterion on error detection*, IEEE Transactions on Information Theory **35** (1989), no. 6, 1299–1305.

[Lei92] F. Thomson Leighton, *Introduction to parallel algorithms and architectures: array, trees, hypercubes*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

[LL12] Feng-Hao Liu and Anna Lysyanskaya, *Tamper and leakage resilience in the split-state model*, CRYPTO, 2012, pp. 517–532.

[NN93] Joseph Naor and Moni Naor, *Small-bias probability spaces: Efficient constructions and applications*, SIAM J. Comput. **22** (1993), no. 4, 838–856.

[NP06] Moni Naor and Benny Pinkas, *Oblivious polynomial evaluation*, SIAM J. Comput. **35** (2006), no. 5, 1254–1281.

[Pip85] Nicholas Pippenger, *On networks of noisy gates*, FOCS, 1985, pp. 30–38.

[RB89] Tal Rabin and Michael Ben-Or, *Verifiable secret sharing and multiparty protocols with honest majority (extended abstract)*, STOC, 1989, pp. 73–85.

[Sha79] Adi Shamir, *How to share a secret*, Commun. ACM **22** (1979), no. 11, 612–613.

[SY10] Amir Shpilka and Amir Yehudayoff, *Arithmetic circuits: A survey of recent results and open questions*, Foundations and Trends in Theoretical Computer Science **5** (2010), no. 3-4, 207–388.

[TS96] Amnon Ta-Shma, *A note on pcp vs. mip*, Inf. Process. Lett. **58** (1996), no. 3, 135–140.

[vN56] J. von Neumann, *Probabilistic logics and synthesis of reliable organisms from unreliable components*, Automata Studies **34** (1956), 43–98.

[WC79] Mark N. Wegman and Larry Carter, *New classes and applications of hash functions*, FOCS, 1979, pp. 175–182.

[WC81] _____, *New hash functions and their use in authentication and set equality*, J. Comput. Syst. Sci. **22** (1981), no. 3, 265–279.

[Yao86] Andrew Chi-Chih Yao, *How to generate and exchange secrets (extended abstract)*, FOCS, 1986, pp. 162–167.