Electronic Colloquium on Computational Complexity, Report No. 76 (2020)

The Round Complexity of Perfect MPC with Active Security and Optimal Resiliency

Benny Applebaum* Eliran Kachlon* Arpita Patra†

Abstract

In STOC 1988, Ben-Or, Goldwasser, and Wigderson (BGW) established an important milestone in the fields of cryptography and distributed computing by showing that every functionality can be computed with perfect (information-theoretic and error-free) security at the presence of an active (aka Byzantine) rushing adversary that controls up to $n/3$ of the parties.

We study the round complexity of *general* secure multiparty computation in the BGW model. Our main result shows that every functionality can be realized in only four rounds of interaction, and that some functionalities cannot be computed in three rounds. This completely settles the round-complexity of perfect actively-secure optimally-resilient MPC, resolving a long line of research.

Our lower-bound is based on a novel round-reduction technique that allows us to lift existing three-round lower-bounds for verifiable secret sharing to four-round lower-bounds for general MPC. To prove the upper-bound, we develop new round-efficient protocols for computing degree-2 functionalities over large fields, and establish the completeness of such functionalities. The latter result extends the recent completeness theorem of Applebaum, Brakerski and Tsabary (TCC 2018, Eurocrypt 2019) that was limited to the binary field.

*Tel-Aviv University, Israel bennyap@post.tau.ac.il, elirn.chalon@gmail.com

†Indian Institute of Science, Bangalore, India arpita@iisc.ac.in

ISSN 1433-8092

1 Introduction

The round complexity of interactive protocols is one of their most important efficiency measures. Consequently, a huge amount of research has been devoted towards characterizing the round complexity of various distributed tasks (e.g., Byzantine agreement [LF82, DR85, FM85], coin flipping [Cle86, MNS16], zero-knowledge proofs [GK96, CKPR01], verifiable secret sharing [GIKR01, KPR10] and secure multiparty computation [Yao86, BMR90, GS18, BL18] under different security models. In this paper, we study the round complexity of *general* secure multiparty computation (MPC) in the classical model of Ben-Or, Goldwasser, and Wigderson [BGW88]. That is, we strive for *perfect (information-theoretic and error-free)* security at the presence of an *active* (aka Byzantine or malicious) rushing adversary that controls up to $n/3$ of the parties.

In more detail, in this setting n parties wish to compute a joint function f of their private inputs. We assume that parties can communicate over *secure point-to-point channels* and, in addition, have an access to a *broadcast* channel. An all-powerful, computationally-unbounded, adversary actively corrupts up to a bounded number t of the parties, and may instruct the corrupted parties to arbitrarily deviate from the protocol. Informally, *perfect* security essentially implies that the honest parties will always get a valid output, and there is a *zero* probability of cheating by the adversary.[1] We focus on the most challenging setting in which the adversary may corrupt up to $t = \lceil n/3 \rceil - 1$ parties, which is known to be the best achievable resiliency threshold in this setting [PSL80, BGW88].

The discovery of perfect MPC is no less than remarkable. It provides everlasting security unconditionally without relying on unproven intractability assumptions. In one of the most seminal results in the area of cryptography and distributed computing, Ben-Or, Goldwasser, and Wigderson [BGW88] and Chaum, Crépeau and Damgård [CCD88] established the following completeness theorem.

Theorem 1.1 (Feasibility of perfect MPC with active security)**.** *Every n-party functionality can be computed with perfect security against a computationally-unbounded adversary that actively corrupts up to $t < n/3$ of the parties.*

The round complexity of information-theoretic MPC was extensively studied [BB89, BFKR90, SYY99, IK00, GIKR01, GIKR02, IK02, PCRR09, IKP10, KPR10, IKKP15, ABT18, ACGJ18, GIS18, ACGJ19, ABT19]. While it is known that perfect actively-secure MPC can be carried in a constant number of rounds [IK02], the exact complexity has remained open. In this paper, we resolve this question, and completely characterize the round complexity of perfect actively-secure MPC.

Theorem 1.2 (Round Complexity of perfect actively-secure MPC)**.** *Four rounds are necessary and sufficient for general MPC with perfect active-security and optimal resiliency of $t < n/3$.*

To prove the theorem, we develop new lower-bound and upper-bound techniques. Before providing a detailed account of our results and techniques, let us describe some of the most relevant previous results regarding the round-complexity of perfectly-secure MPC.

[1] Apart from being a natural goal, perfect security provides important and useful security advantages over protocols that have a negligible probability of failure, see, e.g., [KLR06].

1.1 Previous Works

The classical approach for perfectly-secure MPC (e.g., Theorem 1.1) suffers from a large round complexity. Loosely speaking, the idea is to represent the function f as an arithmetic circuit C over some moderate-size finite field $\mathbb{F}$ of cardinality larger than n, and to distributively evaluate this circuit in a gate by gate manner. At the beginning, each party uses a *verifiable secret sharing* (VSS) sub-protocol [CGMA85] in order to share her input based on degree-t polynomials [Sha79]. The parties then evaluate the circuit over their shares. Addition is performed locally with no interaction, whereas multiplication increases the degree of the underlying secret-sharing polynomial. To compensate this, after every level of multiplication gates the parties employ a "degree-reduction" step that is based again on VSS. Finally, the parties broadcast their shares for the output wires. The resulting round complexity is therefore $(d+1) \cdot R_{\text{vss}} + 1$ where d is the multiplicative depth of C and R_{vss} is the round complexity of the sharing a secret via perfect VSS with optimal resiliency.[2]

Several works have shown how to securely compute various functionalities in constant number of rounds and with small error probability (e.g., [BB89, CD01]). In [IK00, IK02], Ishai and Kushilevitz (IK) presented the *randomizing polynomials* methodology and showed that one can securely reduce, in a round-preserving way, the computation of an arbitrary functionality to the computation of a degree-3 functionality in which each output is a degree-3 polynomial in the inputs/randomness of the parties. This technique, combined with the aforementioned protocols, allows to securely compute an arbitrary functionality in a constant number of rounds. The overhead of the reduction is polynomial in the formula-size (or branching program size) of f, and so the resulting protocol has a polynomial complexity only for functions computable in NC^1 or in (counting versions of) log-space. Similar limitations apply to all known information-theoretic constant-round protocols even in the case of statistical security against a passive adversary.

The discovery of constant-degree randomizing polynomials provides a tighter relation between the round complexity of VSS and the round complexity of general functionalities. Motivated by this connection, Gennaro et al. [GIKR01] studied the round-complexity of VSS, and proved that three rounds are necessary and sufficient for (the sharing phase of) perfect VSS with optimal resiliency of $t = \lceil n/3 \rceil - 1$. A large number of follow-up works have established other bounds on the round-complexity of VSS in different models (statistical and computational security), network setting and under more refined metrics (minimizing broadcast rounds) [PCRR09, FGG+06, KKK09, KPR10, BKP11, PR18]. Since the sharing functionality of VSS is by itself an MPC task, this yields a three-round lower-bound for securely computing a general functionality perfectly. The lower bound of 3 rounds is also implied from the result of [GIKR02] for any $t > 1$.

Very recently, Applebaum et al. [ABT18, ABT19], inspired by breakthroughs in computational MPC [GS18, BL18], presented a multiparty-variant of randomizing polynomials and used it to securely reduce any functionality f to the computation of some degree-2 functionality g. Unlike the IK constructions, this new completeness result is inherently limited to the binary field. That is, the target functionality g is a degree-2 mapping over $\mathbb{F}_2$. As a result, g cannot be directly computed

[2] In fact, using more modern preprocessing tricks (e.g., due to [Bea91, DI05]) one can pay only a single round of interaction per multiplication level at the expense of distributing at the beginning shares of many multiplication triples (A_i, B_i, C_i) where A_i, B_i, C_i are random degree-t polynomials for which $A_i(0) \cdot B_i(0) = C_i(0)$. Since such a sharing can be implemented by a degree-2 functionality, this leads to a round complexity of $R_2 + d + 1$, where R_2 is the complexity of computing an arbitrary degree-2 polynomial. One can further collapse the initial sharing and the first level of multiplication into a single degree-2 functionality, that can be performed in parallel to the triple-generation step, and improve the round complexity to $R_2 + (d-1) + 1$. However, these optimizations do not help for quadratic functions which will be our focus here.

via the polynomial-based protocols which require a field $\mathbb{F}$ of size at least $n+1$. This seemingly technical mismatch becomes a real issue in the setting of an active adversary. Naively, one can try to replace g with its "arithmetic" version G, which is a degree-2 functionality over $\mathbb{F}$ that agrees with g over binary inputs. However, a protocol for G does not translate into a protocol for g since the adversary may use non-binary inputs.[3] Applebaum et al. [ABT19] were not able to overcome this difficulty in the general case, but were able to obtain new improved round-complexity upper-bounds for *weaker* security notions (e.g., where honest parties are allowed to "abort") or with *sub-optimal* resiliency threshold. Specifically, they constructed a three-round perfectly-secure protocol against an active adversary with threshold of $t < n/4$. The use of binary quadratic function significantly complicates the protocol and its analysis [ABT19, Section 6.1, Appendix A]. For comparison, a similar protocol for quadratic functionalities over a large field can be trivially constructed based on existing techniques from [GIKR01].

1.2 Our Results

We prove the first four-round lower-bound for perfectly-secure MPC.

Theorem 1.3 (3-rounds are insufficient for perfect-MPC)**.** *Let $n \geq 4$ and $n/3 \geq t \geq n/4$ be positive integers. Then there exists an n-party functionality f that cannot be computed in three rounds with perfect security and t-resiliency.*

As an immediate corollary, we conclude that the three-round protocol of Applebaum et al. [ABT19] that achieves a resiliency-threshold of $\lceil n/4 \rceil - 1$ cannot be improved, and that at least four rounds are necessary in order to achieve an optimal threshold. To prove the theorem, we introduce a new round-reduction technique, and use it to show that the number of rounds needed for general MPC is strictly larger than the number of rounds needed for VSS. (See Section 2.1 for an outline.) We complement Theorem 3.1 by proving the following upper-bound.

Theorem 1.4 (4-rounds are sufficient for perfect-MPC)**.** *Every n-party functionality can be computed in four rounds with perfect active-security and optimal resiliency of $t < n/3$. The complexity of the protocol is polynomial in n and in the formula-size (or branching program size) of f.*

Taken together, Theorems 1.3 and 1.4 completely characterize the round complexity of perfectly-secure computation with optimal resiliency.

As usual in the context of constant-round information-theoretic MPC, our protocol is efficient only for NC^1 or log-space functionalities. Nevertheless, even for general functions, for which our construction is inefficient, the result remains meaningful since the protocol resists computationally unbounded adversaries.

The protocol from Theorem 1.4 is proven to be perfectly-secure via a black-box non-rewinding simulator whose complexity is polynomial in the complexity of the adversary and the formula (or branching program) size of f. These features (which are common in the information-theoretic setting) have several useful corollaries. First, by [KLR06], this implies that the protocol is also universally composable (UC) [Can01], that is, security is preserved even when many arbitrary protocols are run concurrently with our protocol. Moreover, when the protocol is efficiently computable (e.g., for $f \in \mathrm{NC}^1$), we can use public-key encryption to get rid of the private-channel assumption, at

[3] This transformation actually works when the adversary is passive. Indeed, by using this route [ABT18] constructed an optimal two-round MPC with *passive* perfect security at the presence of honest majority.

the expense of degrading perfect UC-security to computational UC-security. The resulting protocol can be implemented in the "authenticated channels model" without making use of any *set-up assumptions*!

The proof of Theorem 1.4 is based on two components. Our first ingredient is an extension of the degree-2 completeness result of [ABT19] to large fields of characteristic 2.

Theorem 1.5 (Completeness of quadratic functions, extending [ABT19])**.** *Let f be an n-party functionality, and let $k \geq 1$ be an integer. Then there exists a non-interactive reduction from the task of securely computing f to the task of computing a degree-2 functionality over the field $\mathbb{F}_{2^k}$. The reduction preserves perfect, statistical and computational active-security and provides optimal resiliency.*

This new completeness result can be used to significantly simplify some of the protocols from [IKP10, ABT19]. We further believe that it will lead to new round-optimal protocols in other settings. The proof is outlined in Section 2.3. (See also Theorem 5.23 for a full statement of the theorem.) Theorem 1.5 is accompanied with a new four-round protocol for degree-2 functionalities.

Theorem 1.6 (Four-round protocol for quadratic functions)**.** *Every n-party degree-2 functionality over a finite field $\mathbb{F}$ of size at least n, can be securely computed in four rounds with perfect active-security and optimal resiliency of $t < n/3$.*

Just like in the case of our lower-bound, the proof of Theorem 1.6 essentially establishes a (tight) relation between the round complexity of MPC and VSS– it shows that MPC for quadratic functions essentially needs only *one* additional round beyond what is needed for VSS. The proof (which is outlined in Section 2.2 and fully appears in Section 4) introduces several novel techniques.

2 Technical Overview

In the following subsections, we outline the main ideas behind the proofs of our results.

2.1 Lower Bound

Our starting point is the intuition that computing a general functionality f, that mixes the inputs of the parties, is a harder task than computing the VSS (sharing phase) functionality whose input is taken from a *single* party. Following this (somewhat vague) intuition, we expect to pay more, in terms of round complexity, for f than for VSS. Since the cost of perfectly-secure VSS with optimal resiliency is 3 rounds [GIKR01], such a reasoning should yield a lower-bound of 4 rounds. A natural way to formalize the above intuition is to show that a k-round protocol for f can be turned into a $(k-1)$-round VSS protocol. We follow this route and present a novel "protocol-chopping" technique. Details follow.

Chopping a protocol. Consider a k-round protocol π for some functionality f whose properties will be specified later. In such a protocol, each party P_i starts with an input x_i and at the end receives an output y_i. Using standard reductions, we may assume, without loss of generality, that only the first round messages make use of the private point-to-point channels, and all other messages are sent over the broadcast channel. Let us run the protocol π and halt the execution after $k-1$ rounds. At this point, each party holds the public broadcast values that were transmitted in the first $k-1$

rounds, together a private view v_i, that consists of the party's private input/randomness and the private incoming messages received at the first round. We collect all this information in a vector T, and analyze the properties of T that are induced by the security of π. For example, the privacy of the protocol clearly implies that local parts of T that are available to a t-size coalition I of "corrupted" parties, do not reveal any non-trivial information about the inputs of other "uncorrupted" parties. (In fact, each such I induces an equivalence relation over realizable T's.) More importantly, the fact that the protocol is secure against an active adversary implies that any k-th round extension of T (i.e., a vector of broadcast values) cannot violate correctness even if an adversary controls an I-subset of the parties. We will use these properties (and others) to argue that, when f is chosen properly, the chopped-down protocol, π', realizes VSS.

Extracting a VSS. Roughly speaking, we let one of the parties D play the role of the dealer, and think of her f-input as the secret s. The other parties will choose their f-inputs uniformly at random. The definition of f will guarantee that, under this choice of inputs, the input/output values of any I-coalition (that does not contain the dealer) perfectly hide the value of s. The chopped-down protocol π' will be used as a sharing protocol where v_i (together with the broadcast values from the first $k-1$ rounds) plays the role of the i-th share of party P_i. At the reconstruction phase, each party broadcasts its share v_i, and the parties essentially emulate the last round of π. Views which are clearly corrupted (e.g., inconsistent with too many other views) are excluded, and the functionality f is defined with sufficient redundancy, so that, even in this case, the secret can still be recovered based on the f-inputs/outputs of sufficiently many parties (and even at the presence of corruptions). We further show that an undetected cheating corresponds to an adversarial behaviour in π and is therefore protected by the security properties of the protocol. As one may expect, the most challenging part is to show that after the sharing phase, the dealer is committed to his input. Roughly, we show that a violation of the commitment property allows the dealer to break the "independence of inputs" property in π and effectively correlates her input with the input of an honest party. (Interestingly, this part strongly relies on the security of π against a rushing adversary.)

The actual implementation of these ideas, including the exact definition of the functionality f, turns to be quite subtle, and the reader is referred to Section 3 for full details. As part of the proof, we provide general tools for analyzing the state T of chopped-down protocols. We believe that these tools and, more generally, our chopping technique, may lead to new lower-bounds in other contexts.

2.2 Computing Degree-2 Functionalities in Four-Rounds

Consider a degree-2 functionality f over some finite field $\mathbb{F}$ of size $|\mathbb{F}| \geq n+1$. For simplicity, assume that the functionality f takes an input $x \in \mathbb{F}$ from P_1, and an input $y \in \mathbb{F}$ from P_2, and delivers the product xy to all the parties.[4]

The classical protocols. The standard approach for computing such a functionality will be roughly as follows. At the sharing phase, P_1 and P_2 secret-share their inputs via VSS. At the end of this phase, party P_i holds the shares $X(i)$ and $Y(i)$ where X and Y are degree-t polynomials whose free coefficients are x and y, respectively. Next, party P_i computes the product, $X(i) \cdot Y(i)$,

[4]This example is somewhat simpler than the actual complete degree-2 functionalities, and it is chosen for ease of presentation.

of its shares which lie on the degree $2t$ polynomial $Z = X \cdot Y$. The degree of this polynomial is too high to allow noisy-interpolation (at the presence of $t = \lceil n/3 \rceil - 1$ corrupted points), and therefore the parties apply a degree-reduction sub-protocol which distributively transforms the shares $Z(i)$ into new shares $\hat{Z}(i)$ that lie over a random degree-t polynomial $\hat{Z}$ whose free coefficient equals to $Z(0)$. The latter step can be reduced to (many parallel calls of) VSS. Finally, the parties reveal their $\hat{Z}$-shares. Using noisy interpolation, one can recover the free coefficient $\hat{Z}(0)$ even if t of the shares are corrupted. (Such a noisy interpolation is possible whenever the number of "honest points", $(n-t)$, is two times larger than the number t of corrupted points.)

Letting $R_{\text{vss}} = 3$ denote the round complexity of VSS, we derive a protocol whose round complexity is $2 \cdot R_{\text{vss}} + 1 = 7$. One may try to start the degree-reduction phase earlier, before the sharing phase terminates, however, some of the honest-party shares become only available at the last round of the VSS, and so it is not clear how to carry-out such an optimization.[5]

A more liberal coding scheme. We take a different route and deviate from the above blueprint. Let us start by relaxing the role of degree-reduction. Recall that standard VSS generates, as a by-product, second-level shares. That is, when x is shared, the i-th party gets a first-level share $X(i)$ and, in addition, gets, for every $j \in [n]$, a share $X(j,i)$ of the j-th first-level share $X(j)$ where $X(j,1), \ldots, X(j,n)$ all lie on a degree-t polynomial. Our first insight is that instead of reducing the degree of the product polynomial Z, it suffices to reduce the degrees of the second-level polynomials. That is, while the product polynomial Z remains a degree-$2t$ (re-randomized) polynomial that holds xy as its free-coefficient, our degree-reduction protocol will generate a degree-t second-level sharing for each $Z(j)$. This means that the shares, $Z(j,1), \ldots, Z(j,n)$, should lie on a degree-t polynomial. Moreover, we require the existence of a public list G of at least $n-t$ "good" parties, such that for every $P_j \in G$ the second-level shares $Z(j,1), \ldots, Z(j,n)$ are at most t-noisy. Once we have such $n-t$ "good" points we can recover the degree-$2t$ polynomial Z using standard interpolation (which is possible since $n - t > 2t$.)[6]

Of course, one should still implement this second-level degree-reduction, and it is not clear why this task is easier than the original one. In short, the difference is that the second-level degree-reduction for a first-level share $Z(j)$ will be lead by the j-th party P_j, and, unlike first-level degree-reduction, our correctness requirements here are relatively modest. When a corrupted P_j misbehaves, we do not care whether the process succeeds as long as this misbehavior is publicly detected. In such a case, we simply remove P_j from the set G.

Second-level degree-reduction. We briefly explain how to implement the second-level degree reduction. Let us assume for now that each party P_j shared, in some preprocessing phase, a triple of degree t polynomials A, B and C with $A(0) \cdot B(0) = C(0)$ such that each party P_i holds $A(i), B(i)$ and $C(i)$. (The round-complexity of the preprocessing phase is ignored for now.) Beaver's well known reduction [Bea91] uses such a multiplicative-triple to obtain a single-round degree-reduction (i.e., to transform shares of degree-t polynomials F and F' into degree-t shares of $F(0) \cdot F'(0)$) that involves a couple of reconstructions.

[5] One could try to use pre-computed secret-shared multiplicative triples [Bea91], however, in order to generate such triples we need a protocol for degree-2 functionalities.

[6] Using the terminology of error-correcting codes, we essentially replace the standard degree-t Reed-Solomon code, $\text{RS}(n,t)$, that tolerates at most $(n-t)/2$ errors by the tensor code $\text{RS}(n,2t) \bigotimes \text{RS}(n,t)$ that tolerates full erasures of at most t columns together with errors in at most $(n-t)/2$ locations in every un-erased column.

We get rid of the preprocessing assumption by presenting a VSS-based *centralized* triple-sharing protocol in which a *single* dealer chooses the polynomials A, B and C. (The fact that the protocol is centralized saves the need for distributed degree-reduction.) The protocol requires 4 rounds where the sharing is completed in the first 3 rounds and the verification of the product relation of the secrets is completed in Round 4.

Earlier guided degree-reduction. At this point, we still face one final problem. The inputs to Beaver-based round reduction, namely the second-level sharing of $X(i)$ and $Y(i)$ are ready only at the end of Round 3 (upon conclusion of VSS instances). Therefore, one has to spend an additional 4-th round to complete the degree-reduction (even if the random multiplication triples were prepared in an offline phase). This leads to a 5-round protocol. To resolve this issue, let us (naively) assume for now that P_i receives the second-level polynomials $X(i,\cdot)$ and $Y(i,\cdot)$ already at the end of Round 2. Since P_i chose the random multiplication polynomials A, B, C by herself, this tuple is also ready at this point. As a result, P_i has enough information to help the parties execute the additional round needed for degree-reduction already at Round 3.

Of course, a corrupt P_i can cheat and mislead the computation leading to a faulty execution of Beaver's trick. To cope with this, we let each party verify, at Round 4, whether P_i's guided degree-reduction is consistent with the actual second-level shares that were finalized in Round 3. If a misbehavior is detected, P_i is excluded from the set G. Yet again, having at least $2t+1$ honest parties ensures that we will have enough values on the main $2t$-degree polynomial Z even after excluding the outcomes of Beaver's trick corresponding to all corrupt P_is.

VSS in 2.5 rounds. To complete the description, we still have to come-up with a 3-round VSS in which the second-level polynomials $X(i)$ and $Y(i)$ are available to P_i already after two rounds. While we do not know how to satisfy this requirement, we show how to tweak the 3-round VSS of [KKK09] in a way that guarantees a slightly weaker property: At the end of Round 2, each party P_i holds some preliminary version $X'(i,\cdot)$ and $Y'(i,\cdot)$ of the second-level polynomials $X(i,\cdot)$ and $Y(i,\cdot)$ with the guarantee that for any honest party P_i, the preliminary polynomials, $X'(i,\cdot)$ and $Y'(i,\cdot)$, either fully agree with the final polynomials, or agree with the final polynomials on some universal set of $t+1$ points that will become public later (in the end of round 3). This property still suffices for the final reconstruction. As a side note, we present a simpler recipe for VSS through a new building block, Weak Commitment Scheme, that offers a relatively simpler instantiation compared to the traditional building block Weak Secret Sharing.

2.3 Completeness of Degree-2 Functionalities over Large Fields

As already mentioned, our starting point is the recent theorem of [ABT18, ABT19] that establishes the completeness of degree-2 functionalities over the binary field. Their proof proceeds in two steps: (1) Convert the target functionality f into a more friendly functionality F whose circuit is based on a protocol for computing f with the desired security guarantees; (2) Use an "optimized version" of the perfectly-secure garbled circuit (GC) of [IK00] in order to reduce F into a degree-2 functionality g. Recall that the standard (non-optimized) GC-based reduction yields a randomized functionality that has degree-1 in the inputs and degree-2 in the randomness, leading to an overall degree of 3. In the optimized version, instead of *jointly sampling* each internal random-bit that is employed by the GC, we carefully *partition* the random bits between the parties and grant each party full control on his part of the randomness. Consequently, this randomness can be locally preprocessed in a way

that simplifies the residual computation into a degree-2 computation. In general, one cannot just partition the randomness between the parties without violating security, however, [ABT18, ABT19] show that such a partitioning can be safely applied since F itself is based on a secure protocol for f. In order to handle the active setting, it is shown that any adversarial deviation at the local-preprocessing stage can be mapped into a cheating strategy against the original protocol F, and so such a behavior can be simulated based on the simulator of F.

Our goal is to obtain a large-field version of this result. For this purpose, one may try to employ arithmetic variants of the perfectly-secure GC constructions. While such variants exist (see [AIK14, App17]), they do not seem to allow for degree-2 reduction. (A straightforward adaptation of the local preprocessing technique to the arithmetic setting either violates security or leads to large degree of $|\mathbb{F}| > 2$.) Another option is try to embed the binary degree-2 functionality into a degree-2 functionality over a larger field. As explained in the introduction, while this is trivial in the passive setting, when the adversary honestly follows the protocol, this transformation fails at the presence of an active adversary that may use non-binary inputs. Hence, one has to add a mechanism that forces binary behavior. Designing such a *non-interactive degree-2* mechanism is a challenging task, especially in our perfect error-free setting. Indeed, our mechanism should not only *detect* non-binary behavior, it should also *correct* the output delivered to honest parties (i.e., deliver a "good" output to the honest parties which is consistent with some binary input for the corrupted parties), and *erase* the output that corresponds to non-binary inputs (so that corrupted parties do not learn an output that is induced by a non-binary input).[7] We do not know how to directly obtain such a mechanism for a general functionality.

Fortunately, the complete $\mathbb{F}_2$-quadratic functionality turns to satisfy several "nice" properties that significantly simplify our task. In short, we make two main observations: (1) We can successfully cope ("correct and erase") with a non-binary x when x is a "public" variable that will be revealed to everyone by the functionality; (2) Loosely speaking, the inputs to the Boolean perfectly-secure GC functionality are either "arithmetic-friendly" or "publicly-available". Let us elaborate on these two points.

Coping with public inputs. Say that the input x is a public variable whose value will be known to everyone. In this case, we can construct an ifBin gadget that releases a key B (for *binary*) if and only if $x \in \{0,1\}$. Similarly, we can construct an $\mathsf{ifnotBin}$ gadget that releases a key A (for *arithmetic*) if and only if $x \notin \{0,1\}$. Moreover, both gadgets can be implemented by quadratic functionalities that output a pair of elements:

$$\mathsf{ifBin}(B, x; R_1, R_2) := \big(x \cdot R_1 + B, (1-x) \cdot R_2 + B\big),$$
$$\mathsf{ifnotBin}(A, x; R_1, R_2) := \big(x \cdot R_1, (1-x) \cdot R_2, R_1 + R_2 + A\big),$$

where the "keys" A and B are field elements and R_1 and R_2 are fresh random field elements. Indeed, if x equals to zero or one, the first or second output of ifBin reveals B, whereas for any other field element $x \notin \{0,1\}$ both outputs of ifBin reveal no information about B since the random elements $x \cdot R_1$ and $(1-x) \cdot R_2$ act as one-time pads. (An analysis of similar flavor can be applied to $\mathsf{ifnotBin}$ as well.) By using these gadgets one can achieve in principle both correction and erasure. For example, if $F(x, y)$ outputs $(x, g(x, y))$ for some degree-2 function g, we can replace it with the arithmetic

[7] Indeed, one can easily notify the parties whether an input x is binary or not, by appending the quadratic "flag" $y = x^2 - x$ to the output of the function. However, "correction" and "erasure" seem to require a degree-2 computation in y which increases the overall degree to 3 or more.

functionality that outputs the tuple $(x, g(x,y) + B, g(0,y) + A)$ together with $\mathsf{ifBin}(B, x; R_1, R_2)$ and $\mathsf{ifnotBin}(A, x; R_1, R_2)$. When a non-binary input x is being used, the information $g(x,y)$ is being erased (since the key B is not released) and the alternative "corrected" output $g(0,y)$ can be computed (since the key A is being released).

Arithmetizing the Boolean GC functionality. The Boolean GC functionality employs two types of random inputs: (a) *keys* that are used for encrypted gate-tables; and (b) *wire masks* that are used for masking the value (aka signal) that propagates over each wire. The former are being used only in degree-1 computations either as the keys to one-time pads or as the content that is being encrypted. Correspondingly, there is no harm in taking these values to be general field elements. The wire masks are in turn more sensitive. Each such mask is locally-manipulated via degree 2 computation by some party who "owns" the wire. More importantly, the results of these preprocessed values is being used as a "selector" s for one of two keys via an expression of the form $sK_1 + (1-s)K_0$. Replacing such a selector with a general non-binary field element is problematic both for privacy and for correctness. (Roughly, arithmetic selectors allow to run the circuit over a linear combination of the inputs.) We therefore have to force these values to be binary.

Wire masks must be private, and so we cannot handle them via the ifBin and $\mathsf{ifnotBin}$ gadgets. The crucial observation is that the *masked signal* of a wire, which is simply the sum of the wire's mask and signal, is actually public. Furthermore, one can tweak the GC construction so that instead of enforcing binary behavior over masks, it suffices to enforce such a behavior on the masked signals. Following this approach, we integrate the gadgets into the garbled table of each gate, while making sure that a malicious non-binary execution leads to an effective choice of zeroes.

Several technicalities still arise, especially for gates that correspond to broadcast in the original protocol F. In particular, in some cases the adversary may apply a non-binary mask to values that are "owned" by different honest parties. The use of binary extension field guarantees that such a non-binary mask will unanimously throw all these binary values outside $\{0,1\}$, and is therefore translated to a single binary broadcast message. The main details of the construction appear in Sections 5.2 and 5.3, following relevant preliminaries from [ABT18] (in Section 5.1). The proof of the completeness theorem appears in Sections 5.4 and 5.5.

3 Lower-Bound for Perfectly-Secure MPC

In this section we prove Theorem 1.3, restated here for the convenience of the reader.

Theorem 3.1 (Theorem 1.3 restated)**.** *Let $n \geq 4$ and $n/3 \geq t \geq n/4$ be positive integers. Then there exists an n-party functionality F which cannot be computed in 3 rounds with perfect security tolerating t malicious corruptions.*

Proof. By a party-partitioning argument (see [Lyn96]), it is enough to prove the theorem for the case $n = 4$ and $t = 1$. We denote the parties by P_1, P_2, P_3 and P_4 and prove the lower-bound for any 4-party functionality that is admissible as defined below.

Definition 3.2 (Admissible functionality)**.** *Let A and B be finite sets. A 4-party functionality*

$$F : A \times \{0,1\} \times \{0,1\} \times \{\perp\} \to \{\perp\} \times B \times \{\perp\} \times B$$

is admissible *if it can be written as* $F(x,s,t,\perp) = (\perp, f_{s,t}(x), \perp, f_{s,t}(x))$*, where the functions* $f_{0,0}, f_{0,1}, f_{1,0}, f_{1,1} : A \to B$ *satisfy the following properties for every* $s,t \in \{0,1\}$*:*

$$f_{s,t} \text{ is injective} \tag{1}$$
$$\mathrm{Im}(f_{s,t}) = B_{s\oplus t} \text{ where } B_0, B_1 \subset B \text{ are disjoint sets} \tag{2}$$
$$f_{s,t}(x) \neq f_{1-s,1-t}(x), \forall x \in A \tag{3}$$
$$(f_{s,t}(x), f_{1-s,t}(x)) \neq (f_{1-s,1-t}(y), f_{s,1-t}(y)), \forall x,y \in A. \tag{4}$$

We demonstrate with an example that admissible 4-party functionalities exist.

Example 3.3. *Let* $A = \mathbb{Z}_5 = \{0,1,2,3,4\}$ *be the group of integers modulo 5. Let* $B = \mathbb{Z}_5 \times \mathbb{Z}_2$ *and let* $B_0 = \mathbb{Z}_5 \times \{0\}$ *and* $B_1 = \mathbb{Z}_5 \times \{1\}$*. Take* $f_{0,0}(x) = (x,0)$*,* $f_{1,1}(x) = (x+1,0)$*,* $f_{0,1} = (x,1)$ *and* $f_{1,0}(x) = (x+2,1)$ *where addition is over* $\mathbb{Z}_5$*. The reader can verify that Eqs. (1)–(4) hold.*

We now prove the theorem by reducing a perfectly-secure, 3-round, 4-party MPC that computes an admissible function F to a perfectly-secure, 4-party VSS protocol with a 2-round sharing phase, both tolerating one corruption. The latter is known to be impossible [GIKR01, Theorem 8 and Lemma 5], leading to a contradiction. The proof of this theorem thus follows from Theorem 3.4 given below. We present the definition of a perfectly-secure VSS in Appendix A.1 and denote an n-party VSS tolerating t corruption as (n,t)-VSS. □

Theorem 3.4. *If there exists a* 3*-round* 4*-party protocol* π *that computes an admissible functionality* F *with perfect security against a single actively-corrupted party, then there exists a* 4*-party VSS with* 2 *rounds in the sharing phase and perfect security against a single actively-corrupted party.*

To get some intuition, think of s as a secret and of P_2 as the dealer, and consider the case where x and t are uniformly chosen. Then, the properties of an admissible F guarantee that (a) the view of every single party $P_i \neq P_2$ does not reveal any information about the secret s and (b) the input/output pair of every three-party coalition $S \subset \{P_1,P_2,P_3,P_4\}$ completely reveals the secret s. Of course, the main challenge is to show that a 3-round protocol π for F induces a 2-round VSS protocol tolerating malicious behaviour, which is known to be impossible [GIKR01]. The following section describes the transformation from π to a VSS π', and the subsequent sections are devoted to the analysis of the VSS. Lastly, we note that our transformation allows us to reduce any k-round MPC computing F to a $(k-1)$-round VSS. To keep the exposition simple, we continue with $k = 3$.

3.1 The Reduction

Let F be an admissible functionality defined via $f_{0,0}, f_{0,1}, f_{1,0}, f_{1,1}$. Assume towards contradiction that there exists a protocol π that computes some F in 3 rounds. By standard use of one-time pads, we may assume without loss of generality that at the first round each party sends a broadcast message and a private message to each other party, and in all the other rounds, each party sends a broadcast message (see [GIKR01, Lemma 2]). Accordingly, we will use the following notation.

Notation and Terminology. The *private view* v_i of party P_i is a tuple $(z, r_i, (a_{j,i})_{j\in[4]\setminus\{i\}})$ that consists of an input z (possibly $\perp$), randomness r_i, and, for every $j \in [4], j \neq i$ the private message $a_{j,i}$ sent from P_j to P_i at the first round. Note that incoming *broadcast* messages are excluded from

the private view.[8] We denote the input of P_i according to v_i by $\text{input}_\pi(v_i)$, and the randomness of P_i according to v_i by $\text{rand}_\pi(v_i)$. For input z and randomness r, we define $\pi_{1,i}(z,r) := ((a_{i,j})_{j\neq i}, a_i)$ to be the tuple of private messages $a_{i,j}$ that P_i has to send to P_j in the first round of π, and the broadcast a_i of P_i in the first round of π. For a view v_i of P_i and broadcasts $\mathbf{a} = (a_1, \ldots, a_4)$ of the first round, we define $\pi_{2,i}(v_i, \mathbf{a}) := b_i$ to be the broadcast of P_i in the second round of π. For broadcasts $\mathbf{b} = (b_1, \ldots b_4)$ of the second round, we define $\pi_{3,i}(v, \mathbf{a}, \mathbf{b}) := c_i$ to be the broadcast of P_i in the third round of π. For broadcasts $\mathbf{c} = (c_1, \ldots, c_4)$ of the third round, we define $\pi_{o,i}(v_i, \mathbf{a}, \mathbf{b}, \mathbf{c}) := y_i$ to be the output of P_i in the protocol π.

We say that a private view v_i of party P_i is *self-consistent* with respect to broadcast values $(\mathbf{a} = (a_j), \mathbf{b} = (b_j))$ of π, if $\text{input}_\pi(v_i)$ is a valid input of P_i, and the broadcast messages (a_i, b_i) of P_i are consistent with its view and with the other broadcasts, i.e., $a_i = \pi_{1,i}(\text{input}_\pi(v_i), \text{rand}_\pi(v_i))$ and $b_i = \pi_{2,i}(v_i, \mathbf{a})$.

We say that a private view v_i of party P_i is *consistent* with a view v_j of party P_j if the private message that P_i sends to P_j in the first round, as defined by $\pi_{1,i}(\text{input}_\pi(v_i), \text{rand}_\pi(v_i))$, is the same as the message that P_j received from P_i according to v_j, and vice versa.

Overview. Consider the following 2-phase protocol π' in which P_2 is the dealer with input s. At the sharing phase, P_1 and P_3 uniformly pick $x \in A$ and $t \in \{0,1\}$ respectively, and then the parties execute the first two rounds of π. During the reconstruction phase, the parties broadcast their private views and subsequently locally emulate the third round and output computation of π. If P_i's private view is not self-consistent or is inconsistent with two other private views, then a flag α_i is raised and P_i is identified to be corrupt. If one of the parties other than dealer P_2 is identified to be corrupt or P_2's view has no inconsistency, then s is safely retrieved from the private view of P_2. The crux of our construction lies in retrieving s when P_2 is identified to be corrupt or has inconsistency with one of the remaining parties. The properties required from $(f_{s,t})_{s,t\in\{0,1\}}$ as illustrated in Eqs. (1)–(4) are leveraged to identify and output the correct and unique secret s.

Protocol π'

Inputs: P_2 holds an input a secret $s \in \{0,1\}$.

Sharing Phase (2 rounds): P_1 samples $x \leftarrow A$ and P_3 samples $t \leftarrow \{0,1\}$ uniformly at random. In addition, each party P_i samples randomness r_i for the protocol π. All parties invoke the first two rounds of π with inputs x, s, t, and $\perp$, and randomness r_1, r_2, r_3 and r_4, respectively. Let $\mathbf{a} = (a_1, \ldots, a_4)$ denote the broadcast messages of the first round of π, and let $\mathbf{b} = (b_1, \ldots, b_4)$ denote the broadcast messages of the second round of π.

Reconstruction Phase (one round): Each P_i broadcasts its private view v_i. Recall that v_i consists only of the input (if any), the randomness r_i, and the private messages received at the first round.

Local Computation: Given the public values $\mathbf{a}, \mathbf{b}$ and (v_1, v_2, v_3, v_4), every party computes the output s as follows.

[8] Our definition of private view is purely syntactic, and does not necessarily correspond to an actual invocation of π. Of course, when the protocol is invoked (possibly with some malicious party) it naturally induces private views for the honest parties, and we will be typically interested in this case.

1. For $i \in \{1,2,3,4\}$ do:
   (a) If v_i is *not* self-consistent (with respect to $\mathbf{a}$ and $\mathbf{b}$) or if the view v_i is inconsistent with at least two other views $v_j, v_k, j \neq k$, set $\alpha_i = 1$. Otherwise, set $\alpha_i = 0$.
   (b) Define the broadcast message c_i of P_i in the third round of π as follows. If $\alpha_i = 1$ (party P_i is known to be malicious), set $c_i := 0$. Otherwise (if $\alpha_i = 0$), set $c_i := \pi_{3,i}(v_i, \mathbf{a}, \mathbf{b})$. We denote those broadcast values by $\mathbf{c} = (c_1, \ldots, c_4)$.
2. If $\exists i$ for which $\alpha_i = 1$ then:
   (a) If $\alpha_2 = 1$, compute the output $y_4 := \pi_{o,4}(v_4, \mathbf{a}, \mathbf{b}, \mathbf{c})$ of P_4 in π and output the unique s^* for which $f_{s^*,t}(x) = y_4$, where t is the value that P_3 sampled according to v_3 and x is the value that P_1 sampled according to v_1.
   (b) If $\alpha_i = 1$ for $i \neq 2$, the parties output the value s that appears in v_2.
3. Otherwise ($\alpha_i = 0$ for all i), if the view v_2 is consistent with all other views, extract the input s of P_2 from v_2, and output s. Otherwise, let $y_4 := \pi_{o,4}(v_4, \mathbf{a}, \mathbf{b}, \mathbf{c})$ be the output of P_4 in π, let x be the value that P_1 sampled according to v_1, and let t be the value that P_3 sampled according to v_3. We distinguish between the following cases:
   (a) If v_1 is inconsistent with v_2, output zero if there exists some x^* for which $f_{0,t}(x^*) = y_4$. Otherwise, output one.
   (b) If v_2 is inconsistent with v_3, output the unique s^* for which there exists t^* such that $f_{s^*,t^*}(x) = y_4$.
   (c) If v_2 is inconsistent with v_4, output the unique s^* such that $f_{s^*,t}(x) = y_4$.

Figure 1: Protocol π'

The following lemmas show that π' is a $(4,1)$-VSS protocol.

Lemma 3.5. *Protocol π' is a private $(4,1)$-VSS.*

Lemma 3.6. *Protocol π' is a correct $(4,1)$-VSS.*

Lemma 3.7. *Protocol π' satisfies the commitment property of a $(4,1)$-VSS.*

The privacy proof (that depends only on the sharing phase) appears in Section 3.2. In Section 3.3, we collect some useful facts regarding the sharing phase and use them in Sections 3.4 and 3.5 to establish the correctness and commitment properties.

3.2 Privacy (Proof of Lemma 3.5)

Assume that P_2 is honest and fix any adversary $\mathcal{A}$. If $\mathcal{A}$ corrupts P_1 or P_3, then privacy of π' follows from the security of π. (Since the functionality F delivers no output to P_1 and P_2, their views in π must be independent of the inputs of the other parties.) We can therefore focus on the case where $\mathcal{A}$ corrupts P_4. Denote by $\mathcal{D}_s$ the distribution of all the information that is available to the adversary (aka full view) that consists of his private view (defined by the incoming messages in the first round) together with all the broadcasts in the sharing phase of an execution of $\pi'(s)$. We need to show that the distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ are identical.

Let us denote by $\mathcal{D}_s(x,t)$ the distribution $\mathcal{D}_s$ conditioned on the event that at the first round P_1 and P_3 chose the inputs $x \in A$ and $t \in \{0,1\}$, respectively. Recall that under this fixing of x and t, the sharing phase of $\pi'(s)$ is distributed exactly like the first two rounds of $\pi(x, s, t, \perp)$. The perfect security of π therefore guarantees that $\mathcal{D}_s(s,t)$ reveals nothing on the inputs except for $f_{s,t}(x)$, leading to the following claim.

Claim 3.8. *For any pair of input tuples (x_1, s_1, t_1) and (x_2, s_2, t_2) for which $f_{s_1,t_1}(x_1) = f_{s_2,t_2}(x_2)$, the distribution $\mathcal{D}_{s_1}(x_1, t_1)$ is identical to the distribution $\mathcal{D}_{s_2}(x_2, t_2)$.*

Proof. Let $\mathcal{B}$ be an adversary against π who corrupts P_4 and acts like $\mathcal{A}$ in the first two rounds (that correspond to the sharing phase) and in the third round acts arbitrarily, e.g., broadcasts 0. Let us denote by $\mathcal{D}'_s(x,t)$ the full view of $\mathcal{B}$ in an execution of π with honest P_1, P_2 and P_3 whose inputs are x, s and t respectively. Since $\mathcal{D}_s(x,t)$ can be extracted from $\mathcal{D}'_s(x,t)$ (by removing the messages sent in the last round), it suffices to show that $\mathcal{D}'_{s_1}(x_1,t_1)$ and $\mathcal{D}'_{s_2}(x_2,t_2)$ are identically distributed. Indeed, by perfect privacy, there exists an ideal-model simulator $\mathcal{S}$ that given $f_{s,t}(x)$ perfectly samples $\mathcal{D}'_s(x,t)$. Since $f_{s_1,t_1}(x_1) = f_{s_2,t_2}(x_2)$, we conclude that $\mathcal{D}'_{s_1}(x_1,t_1)$ is distributed identically to $\mathcal{D}'_{s_2}(x_2,t_2)$. □

Define the mapping $\delta : A \times \{0,1\} \to A \times \{0,1\}$ that takes (x,t) to $(x', 1-t)$ where x' is the unique element of A for which $f_{0,t}(x) = f_{1,1-t}(x')$. By Claim 3.8 the distributions $\mathcal{D}_0(x,t)$ and $\mathcal{D}_1(\delta(x,t))$ are identical. Moreover, by Eqs. (1)–(4), the mapping δ forms a bijection. Recalling that $\mathcal{D}_s$ is a uniform mixture of the distributions $\{\mathcal{D}_s(x,t) : x \in A, t \in \{0,1\}\}$ we get that

$$\mathcal{D}_0 = \sum_{x\in A, t\in\{0,1\}} \frac{1}{2|A|}\mathcal{D}_0(x,t) = \sum_{x\in A, t\in\{0,1\}} \frac{1}{2|A|}\mathcal{D}_1(\delta(x,t)) = \sum_{x\in A, t\in\{0,1\}} \frac{1}{2|A|}\mathcal{D}_1(x,t) = \mathcal{D}_1,$$

and the lemma follows. □

3.3 Basic Properties

In this section we relate some of the properties of π' to those of π. Except for Claim 3.16, the observations in this section do not make use of the concrete properties of the functionality F. Accordingly, we use a general notation and let z_i denote the input that P_i sends to F and let F_i denote the output that F sends to P_i. (In our case, $z_1 = x \in A$, $z_2 = s \in \{0,1\}, z_3 = t \in \{0,1\}$, $z_4 = \perp$, $F_2 = F_4 = f_{s,t}(x) \in B$ and $F_1 = F_3 = \perp$.)

Recall that once the first step of the reconstruction protocol ends, all the parties agree on a *reconstruction transcript* (RT) $T = (\mathbf{v}, \mathbf{a}, \mathbf{b})$ where $\mathbf{v} = (v_i)_{i\in[4]}, \mathbf{a} = (a_i)_{i\in[4]}$ and $\mathbf{b} = (b_i)_{i\in[4]}$. By applying the local computation step to this vector T, we can define all the intermediate values (e.g., $(\alpha_i, c_i)_{i\in[4]}, y_4$). We refer to these values as the values *induced by* T, and sometimes, when the context is clear, we just treat them as part of the vector T. We use the notation $\text{input}_{\pi'}(v_i)$ and $\text{rand}_{\pi'}(v_i)$ to denote the input and randomness of P_i according to v_i in the simulation of π in π'.

Definition 3.9 (i-realizable in π')**.** *We say that an RT T is i-realizable in π' with respect to inputs $(z_j, r_j)_{j\neq i\in[4]}$ if the transcript can be generated by an adversary P_i^* that corrupts P_i when interacting with the honest parties whose inputs/randomness in the emulation of π are $(z_j, r_j)_{j\in[4], j\neq i}$.*

Clearly, if T is generated by an execution of π then the views of honest parties are self-consistent and consistent with each other. Recalling that the flag α_i is raised if P_i is either not self-consistent or inconsistent with at least two parties, we get the following observation.

Observation 3.10. *Suppose that the RT T is i-realizable in π' with respect to inputs $(z_j, r_j)_{j\in[4], j\neq i}$. Then, exactly one the following holds for the malicious party:*

1. *(detection) $\alpha_i = 1$;*
2. *(single inconsistency) $\alpha_i = 0$ and there exists a single inconsistency between v_i and some v_j.*

3. (consistent transcript) $\alpha_i = 0$ *and there are no self-inconsistencies or pairwise inconsistencies.*

In addition, for every honest party P_j*, the values induced by* T *satisfy* $\alpha_j = 0$ *and* $(\text{input}_{\pi'}(v_i), \text{rand}_{\pi'}(v_i)) = (z_j, r_j)$.

By Observation 3.10, if a transcript T is i-realizable in π' then we do not have to specify the inputs and randomness of the honest parties for which the transcript is generated since these are already defined by T. In order to analyze the correctness and commitment properties, we will have to relate adversarial behavior in π' to adversarial behavior in π.

Definition 3.11 (i-realizable in π)**.** *We say that an RT* T *is* i-realizable *in* π *if there exists an adversary* P_i^* *such that in an execution of* π *in which* P_i *is corrupted by* P_i^* *and the other parties* $P_j, j \neq i$ *play honestly with inputs/randomness that correspond to the* T*-induced values,* $(z_j, r_j)_{j\in[4], j\neq i}$*, the following hold: the view of each honest party* P_j *in* π *is* v_j*, the broadcasts of the first round of* π *are* $\mathbf{a}$*, the broadcasts of the second round of* π *are* $\mathbf{b}$*, and the broadcasts of the third round of* π *are* $\mathbf{c}$*, where* $v_j, \mathbf{a}, \mathbf{b}, \mathbf{c}$ *are computed according to the transcript* T*.*

The following observation follows from the perfect correctness of π.

Observation 3.12. *Suppose that the RT* T *is* i*-realizable in* π *and let* $\mathbf{v}, \mathbf{a}, \mathbf{b}, \mathbf{c}$ *be the values included in and induced by* T*. Then, there exists some input* z_i' *such that for every honest party* P_j *it holds that* $\pi_{o,j}(v_j, \mathbf{a}, \mathbf{b}, \mathbf{c}) = F_j(\mathbf{z}')$*, where the* i*-th entry of* $\mathbf{z}'$ *is* z_i' *and, for* $j \neq i$*, the* j*-th entry of* $\mathbf{z}'$ *equals to the input of* P_j *in* T*.*

It can be shown that if T is i-realizable in π' then it is also i-realizable in π. The proof is deferred to Appendix A.2. Next, we use the following claim that deals with a RT that has a single inconsistency between v_i and v_j (hereafter referred to as (i,j)-*inconsistency*) and each of its views is self-consistent. We refer to such an RT as an (i,j)-*almost consistent RT*.

Claim 3.13 (symmetry of almost-consistent RTs)**.** *Every* (i,j)*-almost consistent RT* T *is both* i*-realizable and* j*-realizable in* π*.*

Proof. Since (i,j)-almost consistent RT is also (j,i)-almost consistent RT, it suffices to prove that T is i-realizable in π. Consider an execution of π in which every party $P_k \neq P_i$, plays honestly with the input/randomness (z_k, r_k) that are taken from T, and the adversary corrupts P_i and behaves as follows. The adversary behaves honestly according to the input/randomness in v_i (and according to the messages that he receives), except that, at the first round, he sends to P_j the private message $a_{i,j}$ that appears in T (as part of P_j's view). We claim that such an execution realizes T.

First, since each view is self-consistent the first-round broadcast in the execution agree with $\mathbf{a}$. Next, observe that in the first-round of the execution, every honest party P_k receives a private message from a party P_ℓ whose value is exactly $a_{\ell,k}$. Indeed, for $(\ell, k) = (i, j)$ this holds by definition, and for all other $(\ell, k) \neq (i, j)$ this holds since the corresponding views are not in a conflict. All the remaining values are consistent with the transcript T since each view is self-consistent with T. □

Definition 3.14 (i-equivalence and i-siblings)**.** *The* i*-part of a transcript* T *consists of* v_i*,* $\mathbf{a}, \mathbf{b}$ *and the induced value* $\mathbf{c}$*. (Intuitively, this is all the information available to a party* P_i*.) In the following we say that* $T = (\mathbf{v}, \mathbf{a}, \mathbf{b})$ *is* i-equivalent *to* $T' = (\mathbf{v}', \mathbf{a}, \mathbf{b})$ *(denoted* $T =_i T'$*) if their* i*-parts are equal. The transcript* T *and* T' *may disagree on private inputs/randomness of party* P_j *for* $j \neq i$*, and on the private messages sent from* P_k *to* P_j *for* $k, j \neq i$*. We say that a pair of input tuple*

$\mathbf{z} = (z_1 \ldots, z_4)$ *and* $\mathbf{z}' = (z'_1 \ldots, z'_4)$ *are* i-siblings *if* $F(\mathbf{z}) = F(\mathbf{z}')$ *and where* $z_i = z'_i$. *(Intuitively, such a pair is indistinguishable from the point of view of an adversary* P_i *that attacks the ideal model by sending* z_i*.)*

The following claim follows from the perfect privacy of π.

Claim 3.15 (sibling transcripts)**.** *Suppose that* T *is* i*-realizable in* π *and let* $\mathbf{z} = (z_1 \ldots, z_4)$ *denote the inputs of the parties according to* T*. Then, for any* i*-sibling* $\mathbf{z}' = (z'_1 \ldots, z'_4)$ *of* $\mathbf{z}$*, there exists an* i*-realizable RT* T' *whose inputs are* $\mathbf{z}'$ *for which* $T =_i T'$*. Moreover, if* T *is* (i, j)*-almost consistent then so is* T' *and if the output of* P_i *in* π *according to* T *is well defined, then it is equal to its output according to* T'*.*

Proof. Let P_i^* be a π-adversary that realizes T. Consider the the following executions of π with P_i^*: (1) The inputs of the honest party are chosen according to $\mathbf{z}$ and their randomness is chosen uniformly; and (2) The inputs of the honest party are chosen according to $\mathbf{z}'$ and their randomness is chosen uniformly.

Let $\mathcal{D}, \mathcal{D}'$ denote the distribution of the complete view of P_i^* in experiment (1) and (2) respectively. That is, $\mathcal{D}$ (or $\mathcal{D}'$) consists of all the incoming messages that P_i^* sees including broadcast values. By perfect security of π, $\mathcal{D}$ is identically distributed to $\mathcal{D}'$. Since the transcript T is generated by P_i^*, the i-part of T is in the support of $\mathcal{D}$. It follows that in $\mathcal{D}'$, there exists a fixing $(r'_j)_{j \neq i}$ of the honest parties randomness, that generates a transcript T' with the i-part of T (that is, $T' =_i T$). We can therefore define T' to be the corresponding transcript.

The "moreover" part, actually holds for every pair of i-realizable transcripts T, T' which are i-equivalent. Indeed, i-realizability implies that the party P_j, $j \neq i$ is honest and so its view v_j is self-consistent with T (resp., T'). Also, the only pair-wise inconsistencies must be with v_i, and since $T =_i T'$, any (i, j) inconsistency in T must holds in T' and vice versa. Finally, since $T =_i T'$, the π-output of P_i must be equal to its output according to T'. □

Finally, the following claim will be used extensively in the correctness and commitment proofs.

Claim 3.16. *For any* $(2, i)$*-almost consistent transcript* T*, it holds that* $\pi_{o,2}(v_2, \mathbf{a}, \mathbf{b}, \mathbf{c}) = \pi_{o,4}(v_4, \mathbf{a}, \mathbf{b}, \mathbf{c}) = f_{s,t}(x)$*, where* x, s *and* t *are the inputs according to* T*, and* $\mathbf{a}, \mathbf{b}, \mathbf{c}$ *are the broadcast values according to or induced by* T*.*

Proof. We split into two cases.

- If $i \in \{1, 3\}$, then T is i-realizable (Claim 3.13), and we can think of P_2 and P_4 as honest parties. Therefore, by Observation 3.12, it holds that $\pi_{o,2}(v_2, \mathbf{a}, \mathbf{b}, \mathbf{c}) = \pi_{o,4}(v_4, \mathbf{a}, \mathbf{b}, \mathbf{c}) = f_{s,t^*}(x^*)$, where either $t^* = t$ (if $i = 1$) or $x^* = x$ (if $i = 3$). On the other hand, since T is 2-realizable, $\pi_{o,4}(v_4, \mathbf{a}, \mathbf{b}, \mathbf{c}) = f_{s^*,t}(x)$ for some s^*. If $i = 1$, then the outputs of P_2 and P_4 equal to both $f_{s,t}(x^*)$ and $f_{s^*,t}(x)$ for some x^*, s^*. This implies that $s = s^*$ since $\mathrm{Im}(f_{s,t}) \cap \mathrm{Im}(f_{1-s,t}) = \varnothing$ and $x = x^*$ since the function $f_{s,t}$ is injective. Similarly, if $i = 3$, then the outputs of P_2 and P_4 equal to both $f_{s,t^*}(x)$ and $f_{s^*,t}(x)$ for some s^*, t^*. It is not hard to verify that Eqs. (1)–(4)) imply that this can happen only when $s^* = s$ and $t^* = t$. In summary, the properties of the functionality F guarantee that $s^* = s$, $x^* = x$ and $t^* = t$.
- If $i = 4$, then T is 4-realizable, by Observation 3.12, it holds that $\pi_{o,2}(v_2, \mathbf{a}, \mathbf{b}, \mathbf{c}) = f_{s,t}(x)$. We now show that $\pi_{o,4}(v_4, \mathbf{a}, \mathbf{b}, \mathbf{c}) = f_{s,t}(x)$. By Claim 3.13, T is 2-realizable in π (i.e., we can think of P_2 as the malicious party). Therefore, by Observation 3.12, the output y_4 of the honest party

P_4 is well defined, and equals to $f_{s^*,t}(x)$ for some s^*. Let us assume, towards a contradiction, that $s^* = 1 - s$ and so $y_4 = f_{1-s,t}(x)$.

First, we use Claim 3.13 again, this time to conclude that T is 4-realizable in π. Recall that $(x, s, t, \perp)$ are the π-inputs in T, and let $x' \in A$ be an input that satisfies

$$f_{s,t}(x) = f_{1-s,1-t}(x').$$

(The existence of such an input follows by the fact that $\mathrm{Im}(f_{s,t}) = \mathrm{Im}(f_{1-s,1-t})$.) The input tuple $(x', 1 - s, t' = 1 - t, \perp)$ is therefore a 4-sibling of the input $(x, s, t, \perp)$, and therefore, by Claim 3.15, there exists a corresponding $(2, 4)$-almost consistent transcript T' in which the inputs are $(x', 1 - s, t' = 1 - t, \perp)$, and for which the induced output y'_4 of P_4 equals to the output of T i.e. $y_4 = f_{1-s,t}(x)$.

Since the transcript T' is $(2, 4)$-almost consistent, it is also 2-realizable (by Claim 3.13), and therefore, by Observation 3.12, the output y'_4 of the "honest" party P_4, must be equal to $f_{s^*,1-t}(x')$ for some s^*. It follows that $y_4 = f_{1-s,t}(x) = f_{s^*,1-t}(x')$, for some $s^* \in \{0, 1\}$. We show that this is impossible, and derive a contradiction. Indeed, if $s^* = 1 - s$, then $f_{1-s,t}(x) = f_{1-s,1-t}(x')$ cannot hold since $\mathrm{Im}(f_{1-s,t}) \cap \mathrm{Im}(f_{1-s,1-t}) = \varnothing$. On the other hand, if $s^* = s$, we recall that x' satisfies $f_{s,t}(x) = f_{1-s,1-t}(x')$, and therefore, by Property (4) of the $f_{a,b}$ functions, $f_{1-s,t}(x) = f_{s,1-t}(x')$ cannot hold.

The claim follows. □

3.4 Correctness (Proof of Lemma 3.6)

Consider an execution of $\pi'(s)$ in which P_2 is honest and some other party P_i is malicious. Let us denote the resulting RT by $T = (\mathbf{v}, \mathbf{a}, \mathbf{b})$ where $\mathbf{v} = (v_i)_{i\in[4]}, \mathbf{a} = (a_i)_{i\in[4]}$ and $\mathbf{b} = (b_i)_{i\in[4]}$. Let $r_j := \mathrm{rand}_{\pi'}(v_j), x := \mathrm{input}_{\pi'}(v_1), t := \mathrm{input}_{\pi'}(v_3)$ and $s := \mathrm{input}_{\pi'}(v_2)$ be the π-inputs induced by T and let $\mathbf{c} = (c_1, \ldots, c_4)$ and $(y_1, \ldots, y_4)$ be the third-round broadcasts and outputs that are induced by T. We will show that the output of the reconstruction phase on T must be s.

By definition, T is i-realizable in π' (and hence in π by Claim A.2). Therefore, by Observation 3.10, all the flags $\alpha_j, j \neq i$ of the honest parties (induced by T) equal to zero including α_2. If the adversary P_i's flag α_i equals to 1, or if P_2 is consistent with everyone else, then the honest parties simply output s from v_2, and we are done. Otherwise, by Observation 3.10, all views are self-consistent and P_2 is inconsistent with P_i, and this is the only inconsistency. Namely, T is $(2, i)$-almost consistent. We distinguish between the following three cases.

P_2 is inconsistent with P_1. In this case, the output of the reconstruction (Step 3a) is 0 iff $y_4 \in \mathrm{Im}(f_{0,t})$ where y_4 is the output of the honest party P_4, induced by T. We will show that the output of the reconstruction equals to s. Since $\mathrm{Im}(f_{0,t}) \cap \mathrm{Im}(f_{1,t}) = \varnothing$, it suffices to show that $y_4 \in \mathrm{Im}(f_{s,t})$. This follows from Claim 3.16

P_2 is inconsistent with P_3. In this case, the output of the reconstruction (Step 3b) is the unique s^* for which there exists t^* such that $f_{s^*,t^*}(x) = y_4$. By Claim 3.16, it follows that $y_4 = \pi_{o,4}(v_4, \mathbf{a}, \mathbf{b}, \mathbf{c}) = f_{s,t}(x)$. Note that s is unique since $\mathrm{Im}(f_{s,t}) \cap \mathrm{Im}(f_{1-s,t}) = \varnothing$ and $f_{s,t}(x) \neq f_{1-s,1-t}(x)$. Therefore the output is s, as required.

P_2 is inconsistent with P_4. In this case, the output of the reconstruction (Step 3c) is the unique s^* for which $f_{s^*,t}(x) = y_4$. By Claim 3.16 it follows that $y_4 = \pi_{o,4}(v_4, \mathbf{a}, \mathbf{b}, \mathbf{c}) = f_{s,t}(x)$. Note that s is unique since $\mathrm{Im}(f_{s,t}) \cap \mathrm{Im}(f_{1-s,t}) = \varnothing$.

This completes the proof of correctness. □

3.5 Commitment (Proof of Lemma 3.7)

Assume, towards contradiction, that the commitment property is violated. That is, there exists a malicious P_2^* that, after the execution *of the sharing phase* with honest parties, can broadcast either $v_{2,0}$ or $v_{2,1}$ in the reconstruction phase so that the honest parties will output 0 or 1, respectively. We will show that such adversary P_2^* allows us to attack the protocol π and violate the "independence of inputs" property.

Notation. Let us denote the corresponding 2-realizable RTs by T_0 and T_1. Observe that these RTs agree on the private views, v_j, for every honest P_j with $j \neq 2$. Let us extract from these views the private randomness, $r_j := \mathrm{rand}_{\pi'}(v_j), \forall j \neq 2$, the inputs, $x := \mathrm{input}_{\pi'}(v_1)$ and $t := \mathrm{input}_{\pi'}(v_3)$, and the private messages, $a_{2,1}, a_{2,3}$, and $a_{2,4}$, that P_2 sent at the first round of π'. Being 2-realizable, the RTs T_0 and T_1 also agree on the first-round broadcast values $\mathbf{a} = (a_1, \ldots, a_4)$, the second round broadcast values $\mathbf{b} = (b_1, \ldots, b_4)$ and the transcript-induced third-round broadcasts of the *honest* parties (c_1, c_3, c_4). Let us denote by $v_{2,0}$ the private view of P_2 under T_0 and by and $v_{2,1}$ the private view of P_2 under T_1. These views may differ, and accordingly they may lead to different third-round broadcast values, $c_{2,0}$ and $c_{2,1}$, for P_2. Let $\mathbf{c}_0 := (c_1, c_{2,0}, c_3, c_4)$ denote the T_0-induced third-round broadcast vector, and by $\mathbf{c}_1 := (c_1, c_{2,1}, c_3, c_4)$ the T_1-induced third-round broadcast vector, respectively.

Observe that at least one of the views, $v_{2,0}, v_{2,1}$, must be self-consistent with its transcript, and consistent with at least two of the other three views. Otherwise, the flag α_2 is raised to 1 in both cases and $c_{2,0} = c_{2,1} = 0$, and so the final reconstructed value defined by $y_4 := \pi_{o,4}(v_4, \mathbf{a}, \mathbf{b}, \mathbf{c})$, will be the same for both T_0 and T_1 (leading to no violation of commitment). Fix $s \in \{0, 1\}$ such that $v_{2,s}$ is self-consistent and inconsistent with at most one other party in T_s. We distinguish between the case where T_s is fully consistent and the case where T_s is $(2, i)$-almost consistent. In both cases, we will prove the following lemma that shows violation of "independence of inputs" in π.

Lemma 3.17. *There exists an adversary $\mathcal{B}$ against π for which the following holds. Consider an execution of π in which P_2 is corrupted by $\mathcal{B}$ and the other parties play honestly with inputs x for P_1 and $t' \in \{0, 1\}$ for P_3.*

1. *If $t' = t$ then, with positive probability, the output of P_4 will be $f_{1-s,t}(x)$.*
2. *If $t' = 1 - t$ then, with probability 1, the output of P_4 will be $f_{s,1-t}(x)$.*

The lemma implies that $\mathcal{B}$ can effectively choose its input based on the input t. Formally, we show that $\mathcal{B}$ cannot be perfectly simulated in the ideal model of F computation. Assume towards a contradiction, that there exists an ideal-model adversary $\mathcal{S}$ that corresponds to $\mathcal{B}$. Since the distribution of the output of P_4 in the real model has the same distribution as in the ideal model for any fixing of the inputs of the honest parties, it must be the case that the ideal adversary sends $1-s$ to the ideal functionality with positive probability. Otherwise, by the first part of Lemma 3.17, $\mathcal{S}$ fails to perfectly simulate $\mathcal{B}$ when the honest inputs are x, t. However, this means that when the honest inputs are $x, 1-t$, the simulator $\mathcal{S}$ sends $1-s$ with positive probability, and so the output

of P_4 will be $f_{1-s,1-t}(x) \neq f_{s,1-t}(x)$. By the second part of Lemma 3.17, this means that $\mathcal{S}$ fails to perfectly simulate $\mathcal{B}$ when the honest inputs are x and $1-t$.

We prove Lemma 3.17 in the following subsections where Section 3.5.1 is devoted to the case where $v_{2,s}$ is consistent with all parties, and Section 3.5.2 is devoted to the case where $v_{2,s}$ is inconsistent with one party.

3.5.1 Case 1: $v_{2,s}$ is self-consistent and consistent with all parties

Since $v_{2,s}$ is consistent with all parties, and all other parties are honest, it follows, by the definition of the reconstruction phase, that the output of the honest parties in T_s is $\text{input}_{\pi'}(v_{2,s}) = s$. We define the following adversary $\mathcal{B}$ who corrupts P_2 in the protocol π.

- In the first round, the adversary plays according to $\pi_{1,2}(s, r_2)$, where $r_2 = \text{rand}_{\pi'}(v_{2,s})$.
- In the second round, the adversary $\mathcal{B}$ sends the message $\pi_{2,2}(v_i, \mathbf{a}')$ where v_i denotes its private view and $\mathbf{a}'$ denotes the broadcast values of the first round.
- At the last round, the adversary $\mathcal{B}$ first sees all broadcasts c'_1, c'_3 and c'_4.[9] The adversary then checks what her output y_2 will be if she continues to play honestly. Formally, she computes $c'_2 = \pi_{3,2}(v_i, \mathbf{a}', \mathbf{b}')$, sets $\mathbf{c}' = (c'_1, c'_2, c'_3, c'_4)$ and computes $y_2 := \pi_{o,2}(v_i, \mathbf{a}', \mathbf{b}', \mathbf{c}')$, where $\mathbf{b}'$ denotes the broadcasts of the second round. If the predicted output, y_2, equals to $f_{s,t}(x)$, the adversary broadcasts the message $c_{2,1-s}$ taken from T_{1-s}, and otherwise she plays honestly with $\pi_{3,2}(v_i, \mathbf{a}', \mathbf{b}')$.

Observe that the adversary is well defined. Indeed, in the first two rounds the adversary plays honestly as if it has input s and randomness r_2, and so $\pi_{1,2}(s, r_2), \pi_{2,2}(v_i, \mathbf{a}')$ and $\pi_{3,2}(v_i, \mathbf{a}', \mathbf{b}')$ are well defined. We will prove Lemma 3.17 for the adversary $\mathcal{B}$.

Proof of Lemma 3.17. Consider an honest execution of $\pi(x, s, t, \perp)$ in which the randomness of P_j is chosen to be r_j for every j (exactly as in π'). This event happens with positive probability, and it would generate the RT T_s if $\mathcal{B}$ continues to play honestly. So the output of P_2 will be equal to $f_{s,t}(x)$, i.e., $y_2 := \pi_{o,2}(v_2, \mathbf{a}', \mathbf{b}', \mathbf{c}') = \pi_{o,2}(v_{2,s}, \mathbf{a}, \mathbf{b}, \mathbf{c}_s) = f_{s,t}(x)$. Consequently, $\mathcal{B}$ will broadcast $c_{2,1-s}$ taken from T_{1-s}. It is easy to note that in this case the values $v_4, x, t, \mathbf{a}, \mathbf{b}$ and $\mathbf{c}_{1-s}$ are consistent with T_{1-s}, because T_s and T_{1-s} agree on the views of the honest parties and all the broadcasts except the round-3 broadcast of P_2. We will show in Claim 3.18 below, that the output of P_4 will be $\pi_{o,4}(v_4, \mathbf{a}, \mathbf{b}, \mathbf{c}_{1-s}) = f_{1-s,t}(x)$, as required for the first part of Lemma 3.17.

Next, consider an execution of π with the adversary $\mathcal{B}$ in which the party P_3 has input $1-t$ and party P_1 has input x. If this execution is completed honestly, then, by correctness, the output of P_4 and P_2 will always be $f_{s,1-t}(x)$. Consequently, the adversary $\mathcal{B}$ will predict the value $y_2 = f_{s,1-t}(x)$ which is never equal to $f_{s,t}(x)$ (since $\text{Im}(f_{s,1-t})$ and $\text{Im}(f_{s,t})$ are disjoint), and so $\mathcal{B}$ will continue to play honestly, and the final output of P_4 will be $f_{s,1-t}(x)$, as required. □

Claim 3.18. *It holds that $\pi_{o,4}(v_4, \mathbf{a}, \mathbf{b}, \mathbf{c}_{1-s}) = f_{1-s,t}(x)$, where $v_4, x, t, \mathbf{a}, \mathbf{b}$ and $\mathbf{c}_{1-s}$ are the values induced by T_{1-s}.*

Proof. By definition, the RT T_{1-s} is 2-realizable in π, and therefore, by Observation 3.12, it holds that $\pi_{o,4}(v_4, \mathbf{a}, \mathbf{b}, \mathbf{c}_{1-s}) = f_{s^*,t}(x)$ for some $s^* \in \{0, 1\}$.

If $v_{2,1-s}$ is not self-consistent with T_{1-s} or $v_{2,1-s}$ is inconsistent with some other view in T_{1-s}, then the output of the honest parties in π' is determined by $\pi_{o,4}(v_4, \mathbf{a}, \mathbf{b}, \mathbf{c}_{1-s})$ which must be

[9] Here we rely, for the first time, on the security of π against a *rushing* adversary.

$f_{1-s,t}(x)$ (since T_{1-s} leads to the reconstruction of $1-s$) and so $s^* = 1-s$. This implies that $\pi_{o,4}(v_4, \mathbf{a}, \mathbf{b}, \mathbf{c}_{1-s}) = f_{1-s,t}(x)$.

Otherwise, $v_{2,1-s}$ is self-consistent, and is also consistent with all parties, and so the honest parties output $\text{input}_{\pi'}(v_{2,1-s})$ which, by assumption, must be $1-s$. Consider an honest execution of $\pi(1-s,t,x,\perp)$ and note that if each party P_j picks up randomness r_j, and P_2 picked randomness $\text{rand}_{\pi'}(v_{2,1-s})$, then the view of P_2 will be $v_{2,1-s}$, the view of the honest party P_j, for $j \neq 2$, will be v_j, and the broadcasts would be $\mathbf{a}, \mathbf{b}$ and $\mathbf{c}_{1-s}$. Since the protocol is correct, the output of P_4 would be $\pi_{o,4}(v_4, \mathbf{a}, \mathbf{b}, \mathbf{c}_{1-s}) = f_{1-s,t}(x)$, as required. □

3.5.2 Case 2: $v_{2,s}$ is self-consistent and inconsistent with P_i

In this case, T_s is $(2,i)$-almost consistent and the final output of the reconstruction over the RT T_s is s. Following Claim 3.16, it holds that $y_4 := \pi_{o,4}(v_4, \mathbf{a}, \mathbf{b}, \mathbf{c}_s) = f_{s,t}(x)$, $y_2 := \pi_{o,2}(v_2, \mathbf{a}, \mathbf{b}, \mathbf{c}_s) = f_{s,t}(x)$ and $\text{input}_{\pi'}(v_{2,s}) = s$.

We will prove Lemma 3.17 with respect to the following adversary $\mathcal{B}$ who corrupts P_2. Recall that $a_{i,2}$ (resp., $a_{2,i}$) denotes the first-round message that P_2 received from P_i (resp., sent to P_i) according to T_s and let $r_2 = \text{rand}_{\pi'}(v_{2,s})$ be P_2's randomness according to T_s.

- In the first round, the adversary acts according to $\pi_{1,2}(s, r_2)$, except that $a_{2,i}$ is taken as per T_s. (Note that since $v_{2,s}$ is inconsistent with P_i, the message $a_{2,i}$ may not be consistent with $\pi_{1,2}(s, r_2)$.)
- P_2 collects her current private view $(s, r_2), (a'_{j,2})_{j\neq 2}$ and modifies it by replacing the i-th incoming message $a'_{i,2}$ with the message $a_{i,2}$ taken from T_s. We denote the *modified view* by v_2. Let $\mathbf{a}' = (a'_1, a'_2, a'_3, a'_4)$ denote the broadcasts of the first round. In the second round P_2 broadcasts $\pi_{2,2}(v_2, \mathbf{a}')$.
- Let $\mathbf{b}' = (b'_1, \ldots, b'_4)$ be the broadcasts of the second round. At the third round, after seeing the broadcast c'_1, c'_3, c'_4 the adversary computes $c'_2 = \pi_{3,2}(v_2, \mathbf{a}', \mathbf{b}')$ and a "prediction" $y_2 := \pi_{o,2}(v_2, \mathbf{a}', \mathbf{b}', \mathbf{c}')$, where $\mathbf{c}' = (c'_1, c'_2, c'_3, c'_4)$. If $y_2 = f_{s,t}(x)$ then P_2 broadcasts $c_{2,1-s}$ taken from T_{1-s}. Otherwise P_2 broadcasts c'_2.

Proof of Lemma 3.17. Consider an execution of $\pi(x, s, t, \perp)$ in which $\mathcal{B}$ corrupts P_2 and the randomness of every honest party $P_j, j \neq 2$ is chosen to be r_j. This event happens with positive probability, and so, such an execution generates the RT T_s if $\mathcal{B}$ chooses to act honestly in the third round. From Claim 3.16 it follows that $y_2 = \pi_{o,2}(v_2, \mathbf{a}', \mathbf{b}', \mathbf{c}') = \pi_{o,2}(v_2, \mathbf{a}, \mathbf{b}, \mathbf{c})$ equals to $f_{s,t}(x)$. Consequently, in our execution, $\mathcal{B}$ broadcasts $c_{2,1-s}$ and so the broadcasts are $\mathbf{a}, \mathbf{b}$ and $\mathbf{c}_{1-s}$. By Claim 3.18, we conclude that in this case the output of P_4 will be $\pi_{o,4}(v_4, \mathbf{a}, \mathbf{b}, \mathbf{c}_{1-s}) = f_{1-s,t}(x)$.

We move on and prove the second case of the lemma. Consider an execution of π with the adversary $\mathcal{B}$ in which P_3 has input $t' = 1-t$ and P_1 has input x. Let us denote the RT that is generated by the first two rounds by T. Observe that T is either fully-consistent or $(2,i)$-almost consistent transcript (the latter since T_s was $(2,i)$-almost consistent). In either case, we need to show that the output of P_4 is $f_{s,1-t}(x)$. If T is fully consistent then, by correctness, $y_2 := \pi_{o,2}(v_2, \mathbf{a}', \mathbf{b}', \mathbf{c}') = f_{s,1-t}(x)$. Since P_2 behaves honestly in Round 3, $y_4 := \pi_{o,4}(v_4, \mathbf{a}', \mathbf{b}', \mathbf{c}') = f_{s,1-t}(x)$, as required. The case of almost-consistent transcript, follows from Claim 3.16.

This completes the proof of Lemma 3.17. □

4 Perfectly-secure Degree-2 Computation in Four Rounds

In this section, we prove Theorem 1.6. We begin with a set of definitions.

4.1 Definitions

Definition 4.1. *A value s is said to be* committed *amongst $\mathcal{P}$, denoted as $\lfloor s \rceil$, if there exists a polynomial $f(x)$ of degree at most t with $f(0) = s$ such that every honest party P_i either holds $f(i)$ or $\perp$ and at least $t+1$ honest parties hold non-$\perp$ values.*

Definition 4.2. *A value s is said to be t-*shared *amongst $\mathcal{P}$, denoted as $[s]$, if there exists a polynomial $f(x)$ of degree at most t with $f(0) = s$ such that every honest party P_i holds $f(i)$.*

Definition 4.3. *A value s is said to be* doubly t-shared *amongst $\mathcal{P}$, denoted as $[[s]]$, if there exist polynomials $f(x), \{f_i(x)\}_{i\in\{1,\ldots,n\}}$, all of degree at most t with $f(0) = s$ and $f(i) = f_i(0)$ for $i \in \{1,\ldots,n\}$ such that $f(0), \{f_i(0)\}_{i\in\{1,\ldots,n\}}$ are t-shared via polynomials $f(x), \{f_i(x)\}_{i\in\{1,\ldots,n\}}$ and every honest P_i holds $f_i(x)$.*

Definition 4.4. *A value s is said to be* doubly $2t$-shared *amongst $\mathcal{P}$, denoted as $\langle s \rangle$, if there exist a degree-$2t$ polynomial $f(x)$ and degree-t polynomials $\{f_i(x)\}_{i\in\{1,\ldots,n\}}$ with $f(0) = s$ and $f(i) = f_i(0)$ for every* honest *party P_i such that $\{f_i(0)\}_{i\in\{1,\ldots,n\}}$ are t-shared via polynomials $\{f_i(x)\}_{i\in\{1,\ldots,n\}}$ and every honest P_i holds $f(i)$ and $f_i(x)$.*

In the double secret sharing definitions, the sharings done for the shares of the secret are referred as *second-level* sharings and the shares of the shares are termed as share-shares. The ith share of s is denoted as s_i (the context will make it clear whether the shares correspond to t or $2t$ sharing). The jth share-share of the ith share s_i of s is denoted as s_{ij}.

The sharings $[\cdot], [[\cdot]]$ and $\langle\cdot\rangle$ are linear i.e. local addition of the shares of $[a]$ and $[b]$ results in $[a+b]$ (similarly for the other types of sharing). Furthermore, addition of $\langle a\rangle$ and $[[b]]$ results in $\langle a+b\rangle$.

4.2 The High-level Idea

Our goal is to build a perfectly-secure MPC protocol that can evaluate any n-party degree-2 functionality (over a field larger than n) with optimal round complexity of 4. The following proposition shows that it suffices to focus on a special family of such functions.

Proposition 4.5. *Let f be an n-party functionality that each of its output can be written as a degree-2 polynomial in the inputs over some finite field $\mathbb{F}$. Then, the task of securely-computing f non-interactively reduces to the task of securely-computing a degree-2 functionality g over $\mathbb{F}$ that each of its outputs is of the form*

$$x^\alpha x^\beta + \sum_{j=1}^{n} r^j, \tag{5}$$

where x^α and x^β are the inputs of party P_α and P_β respectively and r^j is an input of party P_j for $j \in \{1,\ldots,n\}$. The reduction preserves active perfect-security and the resiliency threshold. Moreover, the complexity of the the function g (e.g., its formula-size) is polynomial in n and in the total input length of f.

The proof follows immediately from the locality lemma of randomized encodings [AIK04] and the standard transformation from randomized functionalities to deterministic ones.

By Proposition 4.5, it suffices to focus on functionalities whose output can be written as (5). For simplicity, we will discuss computation of one degree-2 term as above. The extension, guaranteeing that the same x values are used across different degree-2 terms, will follow easily.

Traditionally, evaluating a degree-two function would involve secret-sharing the values and multiplying them distributedly. With t corrupt parties in the system, the secret sharing takes the form of t-sharing and the share-wise multiplication results in a non-random $2t$-sharing of the product. The latter is transformed to a t-sharing via degree-reduction and randomization, and lastly the t-shared product is reconstructed robustly to complete degree-two function evaluation. The degree-reduction in each step of multiplication seems necessary to keep the degree inflation in check when a sequence of multiplications needs to be performed. With degree-two functions as the end goal, we ditch full-fledged round-expensive degree-reduction. Rather we settle for generating a randomized *double* $2t$-sharing of the product which enables robust reconstruction via the second-level t-sharings. That is, we perform one-time degree reduction for the second-level sharings alone.

At a high level, the aim of our upper bound, taking 4 rounds, is to compute $\langle x^\alpha x^\beta + \sum_{j=1}^{n} r^j \rangle$ in the first 3 rounds. Denoting $x^\alpha x^\beta + \sum_{j=1}^{n} r^j$ as y, the last round is used to reconstruct the underlying (randomized) $2t$-degree polynomial of $\langle y \rangle$ via robust reconstruction of the n second-level t-sharings and to identify $n-t$ of them that correspond to correct values on the $2t$-degree polynomial. Since $n > 3t$, the $n-t$ points are enough to interpolate the underlying $2t$-degree polynomial and recover y. To generate $\langle y \rangle$ and reconstruct y, we build a series of building blocks– (a) weak commitment scheme (WC) that generates the most primary type of sharing $\lfloor \cdot \rceil$; (b) VSS that uses WC as an building block and generates $[[\cdot]]$-sharing of a party's secret; (c) a triple-sharing protocol that uses VSS as a building block and verifiably generates $[\cdot]$-sharing of a party's triple secrets a, b, c satisfying the multiplication relation $c = ab$. While WC and VSS requires 3 rounds each, the triple sharing protocol requires 4 rounds where the sharing is completed in first 3 rounds and the verification of the product relation of the secrets is completed in Round 4.

With the above tools, we follow the following route at a high level. The VSS is used to generate $[[\cdot]]$-sharing of the secrets x^α, x^β and r^js. A local multiplication over the shares generates a non-random $2t$-sharing of the product $x^\alpha x^\beta$. A double $2t$-sharing of the product is then completed in two steps. First, every party P_i is required to produce an independent triple sharing which is then used to turn the t-sharing of the ith share of x^α and x^β to a t-sharing of their product via Beaver's trick [Bea91]. The correctness of this computation relies on the correctness of the triple which is confirmed in Round 4. With $n-t$ honest parties, at least $2t+1$ t-sharings are correctly computed. Second, the underlying $2t$ degree polynomial holding the product is randomized using a known trick. Assuming both these steps can be concluded in 3 rounds, Round 4 is used to sum up $\langle x^\alpha x^\beta \rangle$ and $[[r^j]]$ (for $j \in \{1, \ldots, n\}$) to obtain $\langle y \rangle$ (thanks to linearity) and reconstruct all the t-sharings (which in turn reveal the values on the randomized $2t$-degree polynomial sharing y) and verifying the correctness of the triple sharing (which identifies the correct points on the $2t$-degree polynomial).

The second step of randomizing the $2t$ degree polynomial requires generating t double t-sharings and can be achieved in 3 rounds using VSS. However, the first step of realizing Beaver's trick brings an additional challenge. The inputs to the Beaver's trick, namely all the t-sharings, themselves are ready only in the end of Round 3 (upon conclusion of VSS instances), leaving no time for the couple of parallel reconstructions needed for Beaver's trick. We resolve this issue by noting that P_i holds

complete information about the triple (for which she herself is the dealer), the tentative ith share of x^α and x^β as well as their tentative share-shares by the end of Round 2 itself. As a result, P_i can guide the reconstruction of the secrets needed for Beaver's trick in Round 3. A corrupt P_i can mislead, leading to a faulty execution of Beaver's trick. Reconstructing these secrets yet again in Round 4 after their t-sharing are finalized in Round 3 and checking against the ones reconstructed in Round 3 (with the aid of P_i), allows us to identify any misbehaviour done by P_i. Yet again, having at least $2t+1$ honest parties ensure that we will have enough values on the $2t$-degree polynomial, excluding the outcomes of Beaver's trick corresponding to all corrupt P_is.

For the degree-2 completeness, we need to output different y values (yet with the same form). To ensure that the same x inputs are used for computation of all the y values, the same secret sharing of the x values needs to be used for computation of $\langle y \rangle$ as above for all y values. With the above high-level idea, we proceed to describe the building blocks and the final protocol.

4.3 Verifiable Secret Sharing

We begin by describing a VSS protocol, tweaking the construction of [KKK09], in which the tentative share and the tentative share-shares are known to a party already at the end of the second round of computation. Either the tentative share or $t+1$ of the tentative share-shares should turn correct (a party need not know which one between the two and which $t+1$ in the latter case before Round 3). This property is essential for reconstructing the values needed for Beaver's trick in the third round of our final 4-round construction and is not offered straightaway by the known VSS protocols including [KKK09]. On the downside, our new VSS protocol uses broadcast in two rounds (Round 2 and 3), compared to one round of broadcast (Round 3) in the VSS of [KKK09].

As a stepping stone towards VSS, we first build a weaker primitive called weak commitment (WC). WC and opening are distributed information-theoretic variant of cryptographic commitment schemes. It also can be viewed as a (weaker) variant of the typical building block of VSS, known as Weak Secret Sharing (WSS). WC has a clean goal of ensuring that– for a unique secret s, at least $t+1$ honest parties must hold the shares of the secret. WSS, on the other hand, ensures that a unique secret must be committed in the sharing phase so that either the secret or $\perp$ will be reconstructed latter during the distributed reconstruction phase. It is noted that a committed secret in WC needs the help of the dealer for its opening, unlike the secret committed in WSS. With a simpler instantiation, weak commitment and opening are sufficient to build a VSS scheme.

Interestingly, our new recipe for 3-round VSS via WC does not demand two rounds of broadcast, if it is not pressed to offer the above property (of making the tentative share and share-share available). In the respective sections below, we remark how Round 2 broadcasts can be avoided following the footsteps of [KKK09].

4.3.1 Weak Commitment.

We start with the definition of WC. The dealer D starts with a polynomial of degree at most t and generates $\lfloor \cdot \rceil$-sharing of its constant term using the input polynomial. For an honest D, WC in fact produces $[\cdot]$-sharing of the constant term. Primitives with similar spirit had been used to build VSS in different network settings [PR18]. We abstract out the need in terms of a functionality $\mathcal{F}_{\mathsf{wc}}$ given in Fig 2 and present the protocol realizing the functionality below. The dealer sends a polynomial $g(x)$ and a set $\mathcal{P}'$, indicating who should receive a share, to the functionality. An honest D will send $g(x)$ of degree at most t and $\mathcal{P}' = \mathcal{P}$. When a corrupt D sends either a polynomial which is of

degree more than t or a set of size less than $n-t$ (denying shares to at least $t+1$ honest parties), all the parties receive $\perp$ from the functionality.

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

$\mathcal{F}_{\mathsf{wc}}$ receives $g(x)$ and a set $\mathcal{P}'$ from $D \in \mathcal{P}$.

- If $g(x)$ has degree more than t or $|\mathcal{P}'| < n-t$, it sends $\perp$ to every P_i.
- Else it sends $g(i)$ to every $P_i \in \mathcal{P}'$ and $\perp$ to everyone else.

Figure 2: Functionality $\mathcal{F}_{\mathsf{wc}}$

At a high level, D, on holding a polynomial $g(x)$ of degree at most t, initiates the protocol by picking a *symmetric* bivariate polynomial $G(x,y)$ of degree t in both variables uniformly at random over $\mathbb{F}$ such that $G(x,0)$ and $G(0,y)$ are the same as the input polynomial $g(x)$ (with change of variable for $G(0,y)$). Following some of the existing WSS/VSS protocols based on bivariate polynomials [KKK09], the protocol goes as follows: D sends $g_i(x) = G(x,i)$ to party P_i and in parallel the parties exchange random pads to be used for pairwise consistency checking of their common shares. When a bivariate polynomial is distributed as above, a pair of parties (P_i, P_j) will hold the common share $G(i,j)$ via their respective polynomials $g_i(x)$ and $g_j(x)$. Namely, $g_i(j) = g_j(i) = G(i,j)$. For any mismatch, say between (P_i, P_j) during padded consistency check in Round 2, the relevant parties and the dealer disclose their version of the disputed common value in Round 3. The parties, in addition, also discloses the pads that they had used. A conflicting pair is defined as the one that agrees on the pad, but disagrees on the common point on $G(x,y)$. A party in a conflicting pair becomes unhappy when its version mismatches with D's version of the common value. Let W denote the set of happy parties and note that two conflicting honest parties cannot belong to W, implying all the honest parties in W are pairwise consistent and together define a unique symmetric bivariate polynomial, say $G'(x,y)$ and an underlying degree t univariate polynomial $g'(x) = G'(x,0)$, the latter of which is taken as D's committed input. The honest parties in W output the constant term of their $g_i(x)$ polynomials received from D as the share of $g'(x)$. The remaining parties who lie outside W sets their share to $\perp$. For an honest D, all the honest parties belong to W and hold non-$\perp$ shares. Protocol wcom is described in Fig. 3, which we prove realizes functionality $\mathcal{F}_{\mathsf{wc}}$ (Lemma 4.6) in Appendix B.1.

Protocol wcom

Inputs: D has input $g(x)$.

Output: The parties output $[g(0)]$ if D is honest and $\lfloor g'(0) \rceil$ otherwise for some $g'(x)$ of degree at most t. The parties output $\perp$, if D is discarded.

R1: D and every party P_i do the following in parallel.

- D chooses a random symmetric bivariate polynomial $G(x,y)$ of degree at most t in each variable such that $G(x,0) = g(x)$. D sends to each P_i the polynomial $g_i(x) = G(x,i)$.
- Each P_i picks a random polynomial $r_i(x)$ of degree at most t and sends $r_{ij} = r_i(j)$ to every P_j.

R2: Each P_i sets its share $s_i = g_i(0)$. Each pair of ordered pair (i, j), the parties P_i and P_j broadcast $m_i(x) = g_i(x) + r_i(x)$ and $m_{ij} = r_{ij} + g_j(i)$ respectively.

R3: For each pair of ordered pair (i, j) such that $m_i(j) \neq m_{ij}$, the parties (P_i, P_j) and D broadcast $(g_i(j), r_i(j))$, $(g_j(i), r_{ij})$ and $G(i, j)$ respectively.

Local Computation: An ordered pair (P_i, P_j) is called *conflicting* pair if $(g_i(j), r_i(j))$ and $(g_j(i), r_{ij})$ broadcasted by P_i and P_j respectively satisfy (a) $r_i(j) = r_{ij}$ and (b) $g_i(j) \neq g_j(i)$. In a *conflicting* pair (P_i, P_j), P_i (respectively P_j) is said to be *unhappy* if $G(i, j)$ broadcasted by D is not equal to $g_i(j)$ broadcasted by P_i. Let W denote the set of happy parties. If $|\mathsf{W}| < n - t$, then D is discarded and W is reset to $\varnothing$. Every $P_i \notin \mathsf{W}$ resets its share s_i to $\perp$.

Figure 3: Protocol wcom

Lemma 4.6. *Protocol* wcom *realises functionality* $\mathcal{F}_{\mathsf{wc}}$ *tolerating a static adversary* $\mathcal{A}$ *corrupting* t *parties, possibly including the dealer* D*.*

While we never need to reconstruct a $\lfloor \cdot \rceil$-shared secret, non-robust reconstruction can be enabled by allowing D to broadcast the committed polynomial and the parties their shares. The D's polynomial is taken as the committed one if $n - t$ parties' share match with it. Clearly an honest D's opened polynomial will be accepted and a non-committed polynomial will always get rejected.

Remark 4.7 (Minimizing broadcast rounds)**.** *We note that the use of broadcast in Round 2 of* wcom *can be avoided. Similar to the steps of WSS in [KKK09], in Round 1,* P_i *sends the pad* $r_i(x)$ *to* D *via private communication in addition to the current steps. In Round 2, for an ordered pair* (P_i, P_j)*,* P_i *sends* $g_i(j)$ *to* P_j*, and* P_j *sends* $g_j(i)$ *to* P_i *and its received pad* $r_i(j)$ *to* D*, all via private communication. In Round 3,* P_i *says 'disagree' when* $g_i(j) \neq g_j(i)$ *and broadcasts* $g_i(j)$ *and pad* $r_i(j)$*. Otherwise,* P_i *broadcasts padded value* $g_i(j) + r_i(j)$*. A similar step is executed by* P_j*.* D *checks if* $r_i(j)$ *sent by* P_i *is same as* $r_i(j)$ *sent by* P_j*. If not, it broadcasts 'not equal' and* $G(i, j)$*. Otherwise, it broadcasts* $G(i, j) + r_i(j)$*. With these changes, the local computation has the same information as the current version of WC and it goes on exactly as is.*

4.3.2 Verifiable Secret Sharing.

VSS allows a dealer to distributively commit to a secret in a way that the committed secret can be recovered robustly in a reconstruction phase. Our VSS protocol vsh allows a dealer D to generate double t-sharing of the constant term of D's input bivariate polynomial $F(x, y)$ of degree at most t and therefore allows robust reconstruction via Read-Solomon (RS) error correction, unlike the weak commitment scheme wcom. The need is abstracted out as a functionality described in Fig 4.

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

$\mathcal{F}_{\mathsf{vsh}}$ receives $F(x, y)$ from $D \in \mathcal{P}$. If $F(x, y)$ is not a symmetric bivariate polynomial of degree less than or equal to t in both x and y, then it replaces $F(x, y)$ with a default choice of such polynomial. Lastly, it sends $f_i(x) = F(x, i)$ to every P_i.

Figure 4: Functionality $\mathcal{F}_{\mathsf{vsh}}$

At a high level, protocol vsh proceeds in the same way as the weak commitment scheme wcom, except that each blinder polynomial is now committed via an instance of wcom. A happy set, V, is formed in the same way. Two conflicting honest parties cannot belong to V, implying all the honest parties in V are pairwise consistent and together define a unique symmetric bivariate polynomial, say $F'(x, y)$ and an underlying degree t univariate polynomial $f'(x) = F'(x, 0)$, the latter of which is taken as D's committed input. A crucial feature that vsh offers by enforcing the W set of every party in V to have an intersection of size at least $n - t$ with V, is that the blinded polynomial of a corrupt party from V is consistent with $F'(x, y)$. This follows from the fact that the shares (pads) that the parties in W receive as a part of wcom remain unchanged, implying $n - 2t \geq t + 1$ of the honest parties in V ensure the consistency of the blinded polynomial of the corrupt party. This feature crucially enables an honest party P_i that lies outside V (in case of a corrupt dealer) to extract out her polynomial $f'_i(x) = F'(x, i)$ and thereby completing the double t-sharing of $f'(0)$. To reconstruct $f'_i(x)$, P_i looks at the blinded polynomial of all the parties in V who kept her happy in their respective weak commitment instances (implying her share did not change). For each such party, the blinded polynomial evaluated at i and subtracted from P_i's share/pad from the underlying wcom instance, allows P_i to recover one value on $f'_i(x)$. All the honest parties in V (which is at least $t + 1$) contribute to one value each, making sure P_i has enough values to reconstruct $f'_i(x)$. A corrupt party in V, being committed to the correct polynomial as per $F'(x, y)$, with respect to the parties in its W set, cannot inject a wrong value. Protocol vsh is now described in Fig. 3 which we prove realizes functionality $\mathcal{F}_{\mathsf{vsh}}$ (Lemma 4.8) in Appendix B.2.

Protocol vsh

Inputs: D has input $F(x, y)$, a symmetric bivariate polynomial of degree at most t.

Output: The parties output $[[F(0, 0)]]$ when D is honest and $[[F'(0, 0)]]$ otherwise where $F'(x, y)$ is a bivariate polynomial of degree at most t.

R1 D and every party P_i do the following in parallel.

- D sends to each P_i the polynomial $f_i(x) = F(x, i)$.
- Each party P_i picks a random polynomial $h_i(x)$ of degree at most t and initiates an instance of wcom, denoted as wcom_i as a dealer with polynomial $h_i(x)$.

R2 Each pair of ordered pair (i, j), the parties P_i and P_j broadcast $p_i(x) = f_i(x) + h_i(x)$ and $p_{ij} = h_{ij} + f_j(i)$ respectively, where h_{ij} is the share of P_j in wcom_i. In parallel, parties execute **R2** of wcom_i for all $i \in \{1, \ldots, n\}$.

R3 For each pair of ordered pair (i, j) such that $p_i(j) \neq p_{ij}$, the parties (P_i, P_j) and D broadcast $(f_i(j), h_i(j))$, $(f_j(i), h_{ij})$ and $F(i, j)$ respectively. In parallel, parties execute **R3** of wcom_i for all $i \in \{1, \ldots, n\}$.

Local Computation An ordered pair (P_i, P_j) is called *conflicting* pair if (a) $h_i(j) = h_{ij}$ and (b) $f_i(j) \neq f_j(i)$, as broadcasted in **R3**. In a *conflicting* pair (P_i, P_j), P_i (respectively P_j) is said to be *unhappy* if $F(i, j)$ broadcasted by D is not equal to $f_i(j)$ broadcasted by P_i. Let V denote the set of happy parties. The parties execute local computation step for every wcom_i for $i \in \{1, \ldots, n\}$. Let W_i denote the set of happy parties in wcom_i. Remove P_j from W_i if $p_i(j) \neq p_{ij}$ and $h_i(j) \neq h_{ij}$. Remove a party P_i from V if $|\mathsf{V} \cap \mathsf{W}_i| < n - t$ or if there exists some j such that $p_i(j)$ that was broadcasted in **R2** is not

equal to $f_i(j) + h_i(j)$ that were broadcasted in **R3**. If $|\mathsf{V}| < n - t$, then discard D and assume a default sharing and reset $\mathsf{V} = \mathcal{P}$. Otherwise, every $P_i \notin \mathsf{V}$ resets polynomial $f_i(x)$ to the degree t polynomial interpolated over the values $\{p_j(i) - h_{ji}\}_{P_j \in \mathsf{V}; P_i \in \mathsf{W}_j}$ (where $p_j(x)$ was broadcasted by P_j in **R2** and P_i has its share h_{ji} from wcom_j). Finally, every P_i outputs $f_i(0)$ and $f_i(x)$.

Figure 5: Protocol vsh

Lemma 4.8. *Protocol* vsh *realises functionality* $\mathcal{F}_{\mathsf{vsh}}$ *tolerating a static adversary* $\mathcal{A}$ *corrupting* t *parties, possibly including the dealer* D*.*

It is easy to note that vsh generates $[[F(0,0)]]$ via the set of polynomials $\big\{F(x,0), \{f_i(x)\}_{i\in\{1,\ldots,n\}}\big\}$. We define *tentative share and share-share* below which are determined by Round 2 itself and are used in the place of final share and share-shares to allow parallelization and maintain the given round constraint in our final construct.

Definition 4.9 (Tentative share and share-share)**.** *We denote* $f_i(0)$ *received by* P_i *in Round 1 as the* i*th* tentative *share of the committed secret, say* s *and denote it by* $\bar{s}_i$*. We denote* $p_j(i) - h_j(i)$ *by* j*th* tentative *share-share of the* i*th share of the committed secret* s *and refer it by* $\bar{s}_{ij}$*.* $s = F(0,0)$ *if* D *is honest, and* $s = F'(0,0)$ *otherwise for some bivariate polynomial* $F'(x,y)$ *of degree at most* t*.*

We state the following observation which can be checked easily and is used crucially in our final construction

Observation 4.10. *The tentative share of a party* P_i *turns to the actual one if* $P_i \in \mathsf{V}$ *in Round 3 and is recomputed via selected tentative share-shares otherwise. The tentative share-share* $\bar{s}_{ij}$ *turns actual one if* $P_j \in \mathsf{V}$ *and* $P_i \in \mathsf{W}_j$ *hold true in Round 3 and are used to compute the actual share of* s *when* $P_i \notin \mathsf{V}$*. Importantly,* P_i *learns all the tentative share-shares in the end of Round 2 itself.*

With an aim to reduce the number of rounds in which broadcast is invoked, the existing VSS of [KKK09] does not broadcast the blinded polynomials in the second round and this bars a party to determine the tentative share-shares by the end of Round 2. Looking ahead this early finding plays a very crucial role for our final 4-round construction.

Remark 4.11 (Minimizing broadcast rounds)**.** *We conclude this section noting that Round 2 broadcasts in* vsh *can be avoided. First, we employ the WC, with broadcast round minimized, as specified in Remark 4.7. Next, similar to [KKK09] and WC, in Round 1, every party sends their chosen pad polynomial* $h_i(x)$ *to* D *via private communication. In Round 2, for an ordered pair* (P_i, P_j)*, the common points are privately exchanged, and* P_j *reports its received pad from* P_i *privately to* D*. Round 2 is now completely devoid of any broadcast call. Round 3 broadcasts are decided similar to Remark 4.7. That is,* P_i *says 'disagree' when* $f_i(j) \neq f_j(i)$ *and broadcasts* $f_i(j)$ *and pad* $h_i(j)$*. Otherwise,* P_i *broadcasts padded value* $p_i(j) = f_i(j) + h_i(j)$*. A similar step is executed by* P_j*.* D *checks if* $h_i(j)$ *sent by* P_i *is same as* $h_i(j)$ *sent by* P_j*. If not, it broadcasts 'not equal' and* $F(i,j)$*. Otherwise, it broadcasts* $F(i,j) + h_i(j)$*. The local computation of* vsh *remains the same. This leads to a somewhat simpler variant of VSS (compared to [KKK09]) that achieves the same (optimal) round complexity together with a single round of broadcast. However, this protocol does not offer the property mentioned above regarding tentative shares and share-shares.*

4.3.3 Reconstruction of $[s]$.

We recall a known protocol to reconstruct a t-shared secret. We define two variants– rec for public reconstruction and rec_j for private reconstruction to party P_j. rec_j is given below and rec can be realized by running n copies of rec_j for every P_j.

Protocol rec_j

Inputs: Each P_i has s_i (alternatively the parties hold $[s]$).

Output: Party P_j outputs s where $s = f(0)$ and $f(x)$ is the polynomial that is consistent with the shares of the honest parties.

R1 Each P_i sends its share s_i to P_j who applies RS error correction to correct t errors and reconstruct $f(x)$ and output $f(0)$.

Figure 6: Protocol rec_j

4.4 Multiplication Triple Sharing

The goal of this protocol is to allow a dealer to share three values (a, b, c) via VSS such that $c = ab$ holds. We abstract out the need in a functionality $\mathcal{F}_{\mathsf{msh}}$ given in Fig 7 and present our protocol subsequently.

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

$\mathcal{F}_{\mathsf{msh}}$ receives $f^a(x), f^b(x), f^c(x)$ from $D \in \mathcal{P}$. If any of the polynomials is not a polynomial of degree less than or equal to t or $f^c(0) \neq f^a(0)f^b(0)$, then it sends $\perp$ to every P_i and $f^a(i), f^b(i), f^c(i)$ otherwise.

Figure 7: Functionality $\mathcal{F}_{\mathsf{msh}}$

Following the idea proposed in [BGW88] and recalled in [AL17], the dealer chooses two polynomials of degree at most t, $f^a(x)$ and $f^b(x)$ with $f^a(0) = a$ and $f^b(0) = b$. It then picks a sequence of t polynomials $f^1(x), \ldots, f^t(x)$, all of degree at most t such that $f^c(x)$ which is equal to $f^a(x)f^b(x) - \sum_{\alpha=1}^{t} x^\alpha f^\alpha(x)$ is a random polynomial of degree at most t with the constant term equalling ab. Both [BGW88, AL17] elucidate the idea of choosing the coefficients of $f^1(x), \ldots, f^t(x)$ in a way that simultaneously cancels out the higher order coefficients and randomizes the remaining coefficients of the product polynomial $f^a(x)f^b(x)$. The dealer hides these $t+3$ polynomials in symmetric bivariate polynomials and invokes $t+3$ instances of VSS. The check for the product relation $c = ab$ is enabled by letting every party P_i verify if $f^a(i)f^b(i) - \sum_{\alpha=1}^{t} x^\alpha f^\alpha(i)$ equals to $f^c(i)$. P_i raises a complaint in Round 2 if the check fails. Further, every P_i that lies outside a common happy set V, for all the $t+3$ instances of VSS is taken to be unhappy with the dealer and its verification is nullified due to the fact that its final shares from the VSS instances get recomputed. Therefore, for such P_i, the equality is verified publicly in Round 4 after reconstruction which is enabled via the second-level t-sharing. If any of the public check fails, the dealer is concluded to be corrupt and $c \neq ab$. Protocol msh is now described in Fig. 8 which we prove realizes functionality $\mathcal{F}_{\mathsf{msh}}$ (Lemma 4.12) in Appendix B.3.

Protocol msh

Inputs: D has inputs (a, b, c) such that $c = ab$.

Output: By the end of **R3**, the parties output $[[a]], [[b]], [[c]]$ or discards D. If D is not discarded, then by the end of **R4**, the parties output $[a], [b], [c]$ where $c = ab$ holds or output $\perp$.

R1 D chooses $t+3$ random polynomials $f^a(x), f^b(x), f^c(x), f^1(x), \ldots, f^t(x)$, each of degree t such that (a) $f^a(0) = a$, $f^b(0) = b$, $f^c(0) = c$ and (b) $f^c(x) = f^a(x)f^b(x) - \sum_{\alpha=1}^{t} x^\alpha f^\alpha(x)$ as discussed in [BGW88, AL17]. D picks $t+3$ symmetric bivariate polynomials $F^a(x,y), F^b(x,y), F^c(x,y), F^1(x,y), \ldots, F^t(x,y)$ with $F^a(x,0) = F^a(0,y) = f^a(x)$, $F^b(x,0) = F^b(0,y) = f^b(x)$, $F^c(x,0) = F^c(0,y) = f^c(x)$ and $F^\alpha(x,0) = F^\alpha(0,y) = f^\alpha(x)$ for every $\alpha \in \{1, \ldots, t\}$. It then runs $t+3$ vsh instances $\{\mathsf{vsh}^a, \mathsf{vsh}^b, \mathsf{vsh}^c, \{\mathsf{vsh}^\alpha\}_{\alpha\in[t]}\}$, with these polynomials as inputs. Let P_i receive $f_i^a(x), f_i^b(x), f_i^c(x), \{f_i^\alpha(x)\}_{\alpha\in\{1,\ldots,t\}}$ from these instances.

R2 The parties run **R2** of $\{\mathsf{vsh}^a, \mathsf{vsh}^b, \mathsf{vsh}^c, \{\mathsf{vsh}^\alpha\}_{\alpha\in[t]}\}$. If $f_i^c(0) = f_i^a(0)f_i^b(0) - \sum_{\alpha=1}^{t} i^\alpha f_i^\alpha(0)$ does not satisfy, then P_i broadcasts its complain.

R3 The parties run **R3** of $\{\mathsf{vsh}^a, \mathsf{vsh}^b, \mathsf{vsh}^c, \{\mathsf{vsh}^\alpha\}_{\alpha\in[t]}\}$. The parties run **Local Computation** of $\{\mathsf{vsh}^a, \mathsf{vsh}^b, \mathsf{vsh}^c, \{\mathsf{vsh}^\alpha\}_{\alpha\in[t]}\}$ to compute $\{\mathsf{V}^a, \mathsf{V}^b, \mathsf{V}^c, \{\mathsf{V}^\alpha\}_{\alpha\in[t]}\}$. If a party P_i complains in **R2**, remove it from all V and then assign $\mathsf{V} = \mathsf{V}^a \cap \mathsf{V}^b \cap \mathsf{V}^c \cap_{\alpha=1}^{t} \mathsf{V}^\alpha$. If $|\mathsf{V}| < n - t$, discard D. Otherwise, every $P_i \notin \mathsf{V}$ in the execution of each instance in $\{\mathsf{vsh}^a, \mathsf{vsh}^b, \mathsf{vsh}^c, \{\mathsf{vsh}^\alpha\}_{\alpha\in[t]}\}$ resets its polynomial and share as done in the **Local computation** of vsh. At this point, we have $[[a]], [[b]], [[c]]$ ready.

R4 For every $P_i \notin \mathsf{V}$, the parties reconstruct $\{f_i^a(0), f_i^b(0), f_i^b(0), \{f_i^\alpha(0)\}_{\alpha\in[t]}\}$ publicly via running $t+3$ instances of rec on $[f_i^a(0)], [f_i^b(0)]$ $[f_i^b(0)], \{[f_i^\alpha(0)]\}_{\alpha\in[t]}$. D is discarded if $f_i^c(0) = f_i^a(0)f_i^b(0) - \sum_{\alpha=1}^{t} i^\alpha f_i^\alpha(0)$ is not satisfied, in which case every P_i outputs $\perp$. Otherwise, parties output $[a], [b], [c]$ ignoring the second-level sharing.

Figure 8: Protocol msh

Lemma 4.12. *Protocol* msh *realises functionality* $\mathcal{F}_{\mathsf{msh}}$ *tolerating a static adversary* $\mathcal{A}$ *corrupting* t *parties, possibly including the dealer* D*.*

4.5 Degree-2 Computation

Here we show how to compute a degree-2 computation of the following form: $y = x^\alpha x^\beta + \sum_{j=1}^{n} r^j$, where x^α and x^β are the inputs of P_α and P_β respectively and r^j is an input of P_j for $j \in \{1, \ldots, n\}$. Assuming y is the output of P_γ alone, it is easy to extend our protocol to one where every P_γ outputs different y (yet having the same form), while ensuring the same x values are used in the computation of all y. This extended computation was proven to be complete for any polynomial-time computation. The goal is abstracted as a functionality $\mathcal{F}_{\mathsf{d2c}}$ below and the protocol appears subsequently for the computation of a single y. We assume the output is given to everyone for simplicity. The functionality can be modified to take a random input from the rightful recipient P_γ and y can be sent out in blinded form using the randomness as the blinder. The realisation of this slightly modified functionality can be obtained relying on the realisation of the below functionality and additionally asking P_γ to run a VSS on a random polynomial (with a uniform random element m^γ in the constant term). The value y is then reconstructed in blinded form to everyone with m^γ as the blinder, which only P_γ can unblind. Thus, we assume y be dispatched to all in $\mathcal{F}_{\mathsf{d2c}}$.

Functionality $\mathcal{F}_{\mathsf{d2c}}$

$\mathcal{F}_{\mathsf{d2c}}$ receives x^α from P_α, x^β from P_β and r^j from $P_j \in \mathcal{P}$. It computes $y = x^\alpha x^\beta + \sum_{j=1}^{n} r^j$ and returns y to every party.

Figure 9: Functionality $\mathcal{F}_{\mathsf{d2c}}$

At a high level, our protocol generates $\langle x^\alpha x^\beta + \sum_{j=1}^{n} r^j \rangle$ from $[[x^\alpha]], [[x^\beta]], \{[[r^j]]\}_{j\in\{1,\ldots,n\}}$ in first 3 rounds and reconstructs the resultant sharing in Round 4 towards P_γ. The major task is to generate $\langle x^\alpha x^\beta \rangle$, as the final sharing can be obtained from this sharing by simply summing up with $\sum_{j=1}^{n}[[r^j]]$, thanks to linearity. Also all the $[[\cdot]]$-sharing can be generated via VSS instances in 3 rounds. Generating $\langle x^\alpha x^\beta \rangle$ from $[[x^\alpha]]$ and $[[x^\beta]]$ involves two tasks– (a) randomizing the $2t$ degree product polynomial sharing $x^\alpha x^\beta$ that is defined from the local product of the shares of x^α and x^β and (b) turning the second-level $2t$ sharings resulted from local product of the share-shares to t-sharing i.e. degree-reduction of the shares of $x^\alpha x^\beta$. The challenge lies in achieving the above tasks in the same 3 rounds which also produces the double t-sharing of the inputs.

The former task is easier to tackle, assuming the latter one is taken care. It needs generating $\langle 0 \rangle$ and adding the same to $\langle x^\alpha x^\beta \rangle$ which ensures the constant term remains unaffected. Since generating $\langle 0 \rangle$ reduces to a bunch of VSS executions and local computations subsequently, the first task is realizable in 3 rounds. We capture the generation of $\langle 0 \rangle$ as a functionality $\mathcal{F}_{\langle 0 \rangle}$ which can be realized by running n instances of VSS, the ith one dealered by P_i with a polynomial of degree t chosen uniformly at random and then extracting t $[[\cdot]]$-sharing from them such that the secrets are privy to the adversary and the underlying polynomials are uniformly random. While we can extract $n - t \geq 2t + 1$ such sharing using standard extraction techniques (such as via Hyper-invertible matrix [BH08]), t are enough to generate a $\langle 0 \rangle$ as follows. Define $F(x, y) = \sum_{\delta=1}^{t} x^\delta F^{(\delta)}(x, y)$, $f(x) = F(x, 0)$ (this is a $2t$ degree polynomial with constant term as 0), $f_i(x) = F(i, y)$ (these are t degree polynomials) where $\{F^{(\delta)}(x, y)\}_{\delta\in\{1,\ldots,t\}}$ are the underlying bivariate polynomials of the extracted $[[\cdot]]$-sharings. The set $\{f(x), f_1(x), \ldots, f_n(x)\}$ defines $\langle 0 \rangle$. This functionality needs to be corruption-aware [AL17] for the construct above to realize it.

Functionality $\mathcal{F}_{\langle 0 \rangle}$

Given a set of parties $I \subset \mathcal{P}$ that are controlled by ideal adversary $\mathcal{A}$, $\mathcal{F}_{\langle 0 \rangle}$ receives $\{s_i\}_{i\in I}$ and $\{s_{ij}\}_{i\in\{1,\ldots,n\};j\in I}$. It picks a random polynomial of degree at most $2t$, $f(x)$, such that– (i) $f(0) = 0$ and (ii) $f(i) = s_i$ for $i \in I$. It further picks a set of random polynomials $\{f_i(x)\}_{i\in\{1,\ldots,n\}}$ of degree at most t such that for each $f_i(x)$– (a) $f_i(0) = f(i)$ and (ii) $f_i(j) = s_{ij}$ for all $j \in I$. It sends $(f(i), f_i(x))$ to every P_i.

Figure 10: Functionality $\mathcal{F}_{\langle 0 \rangle}$

To achieve the latter task, every P_i generates $([a^i], [b^i], [c^i])$ such that (a^i, b^i, c^i) are random and independent of the actual inputs and satisfy $c^i = a^i b^i$. This is done in 3 rounds via VSS instances as in Protocol msh. We then invoke Beaver's trick to transform the t-sharings of ith share of x^α and x^β, x^α_i and x^β_i respectively, to a t-sharing of their product. The correctness of this relies on whether the shared triple is a multiplication triple or not. The correctness of the triple is concluded in Round 4 and the product of P_i's shares are ignored if the check fails. These exclusions do not hinder recovery of $\langle y \rangle$, where the underlying polynomial is of degree $2t$ and at least $2t + 1$ honest parties are available. However, the major obstacle in completing Beaver's trick arises because the

procedure needs interaction in the form of reconstructing values $(x_i^\alpha - a^i)$ and $(x_i^\beta - b^i)$, which seems can be run in Round 4 at the earliest after the input sharings are concluded in Round 3. This drags the preparation of $\langle y \rangle$ to 4 rounds, requiring another round for the reconstruction. We get around the problem by allowing a guided reconstruction of the above values in Round 3 itself, bypassing the traditional reconstruction of a t-shared secret which can be run only after the sharings of $x_i^\alpha, x_i^\beta, a^i, b^i, c^i$ are completed. To be specific, party P_i guides the reconstruction of $(x_i^\alpha - a^i)$ and $(x_i^\beta - b^i)$ on holding the *tentative shares and share-shares* of x^α and x^β in the respective VSS instances and the complete knowledge about its dealt sharings of a^i, b^i. The guided-reconstruction ensures that correct values are reconstructed when the guide P_i is honest, leveraging the conditions under which the tentative shares and share-shares turn the actual ones. A corrupt guide P_i can mislead in Round 3, only to be caught in Round 4 when these values are reconstructed yet again via the standard reconstruction. Leveraging super-honest majority, we can ignore the share y_i for the reconstruction of y. The protocol appears in Fig 11 and the proof that it realizes functionality $\mathcal{F}_{\mathsf{d2c}}$ (Theorem 4.13) in Appendix B.4.

Protocol d2c

Inputs: P_α and P_β input x^α and respectively x^β. In addition, P_j inputs r^j for $j \in \{1, \ldots, n\}$.

Output: Every party outputs $y = x^\alpha x^\beta + \sum_{j=1}^n r^j$.

R1 The parties do the following in parallel

- P_α picks a symmetric bivariate polynomial of degree at most t in each variable $X^\alpha(x, y)$ with $X^\alpha(0, 0) = x^\alpha$ and initiates an instance of vsh, denoted as vsh^α. P_β picks $X^\beta(x, y)$ with $X^\beta(0, 0) = x^\beta$ and initiates an instance of vsh, denoted as vsh^β. Each P_j picks $R^j(x, y)$ with $R^j(0, 0) = r^j$ and initiates an instance of vsh, denoted as vsh^j with input $R^j(x, y)$.
- Every P_i initiates an instance of msh, denoted as msh^i, with inputs (a^i, b^i, c^i), randomly chosen, yet satisfying product relation $c^i = a^i b^i$.
- The parties invoke $\mathcal{F}_{\langle 0 \rangle}$.

R2 The parties run **R2** of $\mathsf{vsh}^\alpha, \mathsf{vsh}^\beta, \{\mathsf{vsh}^i, \mathsf{msh}^i\}_{i \in \{1,\ldots,n\}}$.

R3 P_i broadcasts $\big((\bar{x}_i^\alpha - a^i), \{\bar{x}_{ik}^\alpha - a_k^i\}_{k \in \{1,\ldots,n\}}\big)$ and $\big((\bar{x}_i^\beta - b^i), \{\bar{x}_{ik}^\beta - b_k^i\}_{k \in \{1,\ldots,n\}}\big)$, where $\bar{x}_i^\alpha$ and $\bar{x}_{ik}^\alpha$ denote ith tentative share of x^α and kth tentative share-share of x_i^α (and similarly $\bar{x}_i^\beta$ and $\bar{x}_{ik}^\beta$ are defined). The parties run **R3** and subsequently the local computation steps of $\mathsf{vsh}^\alpha, \mathsf{vsh}^\beta, \{\mathsf{vsh}^i, \mathsf{msh}^i\}_{i \in \{1,\ldots,n\}}$. Let $\mathsf{V}^\alpha, \mathsf{V}^\beta$ and V^i denote the happy sets in the vsh instances with the corresponding superscripts. The W sets inside vsh^α are denoted as W_k^α for $k \in \{1, \ldots, n\}$. Similarly, we denote the W sets inside the remaining VSS instances.

Local Computation

- If both P_α and P_β are discarded, default values of x^α and x^β and default $\langle \cdot \rangle$-sharing of their product is assumed.
- If P_α is discarded, a default value of x^α is assumed and $[[x^\beta]]$ is transformed to $[[x^\alpha x^\beta]]$ via linear transformation. In a similar way, the case when P_β is discarded is taken care. In either case, $[[x^\alpha x^\beta]]$ is taken as $\langle x^\alpha x^\beta \rangle$.

- Otherwise, the ith second-level sharing of $\langle x^\alpha x^\beta \rangle$ is computed as follows. If P_i is discarded, as a dealer, in msh^i, assume a default $[\cdot]$-sharing for the ith second-level sharing and exclude P_i from a set L that is initiated to $\mathcal{P}$. Otherwise, run an instance of Beaver's trick, given $([a^i], [b^i], [c^i])$ and $[x_i^\alpha], [x_i^\beta]$, to locally compute $[x_i^\alpha x_i^\beta]$ as follows:
  - Compute $(x_i^\alpha - a^i)$ as $(\bar{x}_i^\alpha - a^i)$ if $P_i \in \mathsf{V}^\alpha$ and as the constant term of the degree t polynomial interpolated over $\{\bar{x}_{ik}^\alpha - a_k^i\}_{P_k \in \mathsf{V}^\alpha \wedge P_i \in \mathsf{W}_k^\alpha}$ otherwise. If no such t degree polynomial exist, remove P_i from L and a default $[\cdot]$-sharing for the ith second-level sharing is assumed.
  - In a similar way, compute $(x_i^\beta - b^i)$.
  - Compute $[x_i^\alpha x_i^\beta] = (x_i^\alpha - a^i)(x_i^\beta - b^i) + (x_i^\alpha - a^i)[b^i] + (x_i^\beta - b^i)[a^i] + [c^i]$.
- Compute $\langle y \rangle = \langle x^\alpha x^\beta + \sum_{j=1}^n r^j \rangle = \langle x^\alpha x^\beta \rangle + \langle 0 \rangle + \sum_{j=1}^n [[r^j]]$, where $\langle 0 \rangle$ is returned by $\mathcal{F}_{\langle 0 \rangle}$.

R4 For every $P_i \in \mathsf{L}$, the parties run **R4** of msh^i and run two instances of rec for $[x_i^\alpha - a^i]$ and $[x_i^\beta - b^i]$. If P_i is discarded in msh^i or the reconstructed values do not match with the ones used in Beaver's trick in the previous step, remove P_i from L. For every $P_i \in \mathsf{L}$, run an instance of rec on the ith second-level sharing $[y_i]$ of $\langle y \rangle$ to reconstruct y_i. Every party uses the shares to interpolate the $2t$-degree polynomial holding y in the constant term and outputs y.

Figure 11: Protocol $\mathsf{d2c}$

Theorem 4.13. *Protocol* $\mathsf{d2c}$ *realises functionality* $\mathcal{F}_{\mathsf{d2c}}$ *tolerating a static adversary* $\mathcal{A}$ *corrupting* t *parties.*

5 Completeness of Degree-2 Functionalities over Large Fields

Overview. In this section, we prove Theorem 1.5 and securely reduce the computation of a general functionality f to the computation of degree-2 arithmetic functionalities. The reduction will have the following form: first the parties apply some local preprocessing step pre, then the outputs of the preprocessing step will be fed into a degree-2 arithmetic functionality enc whose output will be delivered to all parties, and finaly each party will apply some local post-processing step dec.

As mentioned in the introduction, the reduction is obtained by garbling a protocol Π for f. Following [ABT19], we represent the protocol Π by a Boolean circuit C. Roughly, the reader should envision a protocol as a huge Boolean circuit composed of local computation gates (that belong to a specific party), standard transmission gates (that copy a bit from the output of a local gate of party i to the input of a local gate of party j) and broadcast gates that deliver a bit from party i to all other parties. (See Section 5.1.5 for a formal definition.)

While standard garbling is composed of two procedures: encoding and decoding, we, again follow the outline of [ABT19] and partition the encoding procedure into a preprocessing algorithm pre and an encoding algorithm enc. As the names suggest, these algorithms will be used in the reductions, together with the decoder algorithm dec that will be essentially used as (part of) the post-processing step. In the preprocessing phase, the truth-tables of the gates of C are permuted according to random mask values. Each party chooses the masks for the wires that are controlled by her in the protocol C, and honest parties should use binary masks. The preprocessing function delivers the permuted gates together with the inputs for the input gates (which are again supposed to be binary values).

In the encoding phase, the outputs of the preprocessing algorithm are being used in order to generate the encrypted gate tables. This step uses keys (as well as additional random values) that

are uniformly chosen from the large field $\mathbb{F}$ (taken to be an extension field of the binary field). In each row of a gate table one has to choose which plain-text to encrypt out of two possible options, labeled by zero and one. Since the "selection bit" may be non-binary (due to malicious behavior in the preprocessing step), we implement the selection mechanism via a special "key-selection" gadget select, that will essentially force a binary selection. (A non-binary selection will be translated into a zero-selection.) A more complicated "triple-selection" gadget will be needed in order to handle the case of transmission gates in which a party sends his input to several other parties. The analysis also becomes more complicated, and in order to model a faulty preprocessing we will have to consider a new type of Boolean gates referred to as *generalized transmission* gates. (Notably, this issue does not arise in the binary setting and transmission gates are handled seamlessly.) These modifications, which appear in Sections 5.2 and 5.3, form the main difference compared to [ABT19].

Eventually, we will show that enc can be implemented by a degree-2 mapping, and that the reduction remains secure even in the presence of misbehavior in the preprocessing phase. Roughly, speaking we will show that when an adversary that corrupts a coalition I misbehaves at the preprocessing stage (i.e., sends corrupted inputs and uses corrupted wire masks), the enc function essentially reveals only the information that is released by a circuit $\hat{C}$ in which the gates corresponding to I were modified. Since $\hat{C}$ forms a cheating strategy by I against the protocol Π, the reduction remains secure. Again the argument follows [ABT19] with the technical modifications induced by the need to cope with non-binary inputs. As a side note, we state and analyze the security of our construction (Lemma 5.20) at the level of circuits using the randomized-encoding terminology [IK00, AIK04] without making a direct reference to MPC, we hope that this formulation may turn to be useful in other contexts.

Organization. This section starts with a preliminaries (Section 5.1), setting the stage with the required definitions. It then moves on to describe the key-selection gadgets (Section 5.2) which are used in the construction of the garbled circuit (Section 5.3). Based on this, we derive, in Section 5.4, a general master theorem that reduces general secure computation in various settings (perfect, statistical, computational) into degree-2 arithmetic secure computation. The general completeness theorems are then derived in Section 5.5.

5.1 Preliminaries

In this section, we define Boolean circuits (Section 5.1.1), randomized encoding of functions (Section 5.1.2), multi-party (oracle-aided) protocols (Section 5.1.3), security for multi-party computation (Section 5.1.4) and circuit representation of protocols (Section 5.1.5). Large parts of the texts (and in particular the MPC sections) are taken, with minor changes, from [ABT19].

An expert reader may choose to skip most of the preliminaries, except for Section 5.1.1, while keeping in mind that, by default, we consider an active non-adaptive rushing adversary that is computationally unbounded and assume a fully connected network with point-to-point private channels and a broadcast.

Notation. We denote by $[n]$ the set $\{1,\ldots,n\}$. We denote by $\mathbb{F}_2$ the finite field of size 2, and by $\mathbb{F}$ a finite field of characteristic 2. For any set $S \subseteq [n]$, we denote $\overline{S} = [n]/S$. For any sequence $\mathbf{x} = (x_1,\ldots,x_n)$ and any $S \subseteq [n]$ let $\mathbf{x}[S]$ denote the ordered set $\{x_i\}_{i\in S}$.

5.1.1 Boolean Circuits

In this work, we consider Boolean circuits containing the following types of gates:

- An *input* gate that has no incoming wires and one outgoing wire and is labeled by a unique formal variable X_i. Similarly, an *output* gate has no outgoing wires and a single incoming wire and it is labeled by a unique formal variable Y_i. A wire that is incident to an input gate (resp., output gate) is referred to as an input wire (resp., output wire).

- A *local computation gate* (or local gate in short) has two (ordered) input wires and one output wire. The gate is labeled by some arbitrary function $G : \{0,1\}^2 \to \{0,1\}$ (that can vary from one gate to the other). The value of the outgoing wire is computed by applying G to the values of the input wires.

- A *simple transmission gate* (or transmission gate, in short), that has a single input and an arbitrary number, p, of outputs. The gate simply copies its input to all outputs (and is also referred to as *fan-out* gate).

- A *generalized transmission gate* that has a single input and an arbitrary number, p, of outputs, and is labeled by p pairs of bits $(a_j, b_j)_{j\in[p]} \in (\{0,1\}^2)^p$. On input $\beta \in \{0,1\}$ to the gate, the output of wire j is $a_j \cdot \beta + b_j$, where the computation is over $\mathbb{F}_2$. For example, when $a_1 = \ldots = a_p = 1$ and $b_1 = \ldots = b_p = 0$ the gate becomes a simple transmission gate. (In general, however, different output wires of the gate can have different output values.)

A circuit with n inputs and ℓ outputs computes a function from $\{0,1\}^n$ to $\{0,1\}^\ell$ in the natural way.

Complexity. For purposes of analysis, we define the *depth* of a p-ary simple/generalized transmission gate to be $\lceil \log p \rceil$, and the depth of an input gate or a local gate to be 1. The depth of a circuit C is the computed by considering the cumulative depth of gates along each path from an input wire to an output wire in C, and taking the maximum among all paths. The *size* of a circuit, m, is the number of wires in the circuit (including input and output wires). We assume a topological ordering of the wires in $[m]$.

Two Boolean circuits C and C' are *topologically* equivalent if the directed acyclic graphs that represent the circuit are isomorphic.

Remark 5.1. *We make two simplifying assumptions.*

- *We assume that the fan-out of input-gates and local-gates is always* 1*. This is without loss of generality since we can use fan-out gates in order to increase the fan-out to some constant greater than* 1*, at the cost of multiplying the size and depth of the circuit by a constant.*

- *We assume that transmission gates and generalized transmission gates are only adjacent to local computation gates. This is without loss of generality because we can always insert a dummy local-computation "copy" gates before and after every transmission gate.*

5.1.2 Randomized Encoding of Functions

We use randomize encoding of functions [IK00, AIK04] to replace a high-degree function $f(x)$ with a low-degree randomized function $\hat{f}(x;r)$, whose output reveals only the value $f(x)$ and hides any other information about x.

Definition 5.2 (Randomized encoding)**.** *Let $f : \mathbb{F}^n \to \mathbb{F}^s$ be a function. A function $\hat{f} : \mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F}^t$ (perfectly)* encodes *f is there exists a deterministic algorithm D (decoder) and a randomized algorithm S (simulator) such that for every input $x \in \mathbb{F}^n$, the distribution $\hat{f}(x;r)$ induced by a uniform choice of $r \leftarrow \mathbb{F}^m$, encodes the value $f(x)$ in the following sense:*

- *(Perfect correctness)* $\Pr_r[D(\hat{f}(x;r)) \neq f(x)] = 0$.
- *(Perfect privacy) The simulator $S(f(x))$ perfectly samples the distribution $\hat{f}(x;r)$.*

The randomness complexity *and the* output complexity *of the encoding are m and t, respectively. We say that the encoding has an* arithmetic complexity *of T if $\hat{f}, D$ and S can be computed by making at most T arithmetic operations over $\mathbb{F}$. (For concreteness, we measure T as the size of the corresponding arithmetic circuits; this guarantees that T always upper-bounds m and t.) We say that $\hat{f}$ is of* degree *d if each of its outputs can be written as a degree-d polynomial in the inputs x, r over $\mathbb{F}$.*

Randomized encodings satisfy several useful closure properties that we summarize in the following lemmas (see [AIK04, AIK14, App17] for proofs).

Lemma 5.3 (Composition)**.** *Suppose that $g(x;r_g)$ is an encoding of $f(x)$ with decoder D_g and simulator S_g, and that $h((x,r_g);r_h)$ is an encoding of the function $g(x,r_g)$, viewed as a single-argument function, with decoder D_h and simulator S_h. Then the function $\hat{f}(x;(r_g,r_h)) = h((x,r_g);r_h)$ together with the decoder $D(\cdot) = D_g(D_h(\cdot))$ and the simulator $S(\cdot) = S_h(S_g(\cdot))$ forms an encoding of $f(x)$.*

Lemma 5.4 (Concatenation)**.** *Suppose that, for every $i \in [c]$, the randomized function $\hat{f}_i(x;r_i)$ encodes the function $f_i : \{0,1\}^n \to \{0,1\}^{\ell_i}$ with decoder dec_i and simulator sim_i. Then the function $\hat{f}(x;(r_1,\ldots,r_c)) = (\hat{f}_1(x;r_1),\ldots,\hat{f}_c(x;r_i))$ together with the decoder $D(\hat{y}_1,\ldots,\hat{y}_c) = (D_1(\hat{y}_1),\ldots,D_c(\hat{y}_c))$ and simulator $S(y_1,\ldots,y_c) = (S_1(y_1),\ldots,S_c(y_c))$ encodes the function $f(x) = (f_1(x),\ldots,f_c(x))$.*

Lemma 5.5 (Substitution)**.** *Suppose that the function $\hat{f}(x;r)$ is an encoding of $f(x)$ with decoder D and simulator S. Let $h(z)$ be a function of the form $f(g(z))$ where $z \in \{0,1\}^k$ and $g : \{0,1\}^k \to \{0,1\}^n$. Then, the function $\hat{h}(z;r) = \hat{f}(g(z);r)$ is an encoding of h with the same simulator and the same decoder.*

5.1.3 Functionalities and Protocols

It will be convenient to treat functionalities and protocols as finite (fixed length) objects. The infinite versions of these objects will be defined and discussed latter in Section 5.1.4. We continue with a formal definition.

Definition 5.6 (multi-party functionality)**.** *An n-party functionality $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ is a (possibly randomized) function that maps a sequence of n inputs $\mathbf{x} = (x_1, \ldots, x_n)$ to a sequence of n outputs $\mathbf{y} = (y_1, \ldots, y_n)$. If f sends the same output to all parties then we denote its output as a scalar, i.e. we use the shorthand $f : (\{0,1\}^*)^n \to \{0,1\}^*$ and $y = f(x_1, \ldots, x_n)$.*

Next we define a multi-party protocol in a non-asymptotic setting.

Definition 5.7 (multi-party protocol, oracles)**.** *An n-party, r-round protocol Π is a tuple of $n(r+1)$ simple Boolean circuits $\{C_{j,i}\}_{j\in[r+1],i\in[n]}$ that correspond to the computation that party i in the protocol performs before the j-th communication round (or after the last round if $j = r+1$). Each $C_{j,i}$ (except for $j = 1$ and $j = r+1$, see below) takes n input strings, and outputs n output strings. The i'-th output of $C_{j,i}$ is the message sent from party i to party i' at round j of the protocol. If $i = i'$ then the respective output is the* state *of party i after the j-th round of communication. We therefore require that for all i, i', j the i'-th output of $C_{j,i}$ has the same length as the i-th input string of $C_{j+1,i'}$. In the first round of communication $C_{1,i}$ only takes one input x_i to be interpreted as party i-th input for the protocol, and possibly an additional random string. In the last round of communication $C_{r+1,i}$ only has one output which should be interpreted as the output of party i in the protocol, sometimes denoted y_i. We let M_i denote the collection of circuits associated with party i, i.e. $M_i = (C_{1,i}, \ldots, C_{r+1,i})$ and thus denote $\Pi = (M_1, \ldots, M_n)$. The view of the party in the protocol contains its input, randomness and all messages it received during the execution.*

Let h be an n-party functionality. A protocol Π with oracle h, which we denote by Π^h, is one that allows to replace some of the communication rounds with calls to the functionality h (i.e. the circuits respective to this round each produce one output that is sent to the oracle h as input, the outputs of h is then fed as a single input to the next round circuit).

A protocol with broadcast is one with access to the broadcast functionality that on input $\mathbf{x} = (x_1, \ldots, x_n)$ outputs $\mathbf{x}$ to all parties. More generally, the framework in this paper can handle any oracle functionality that delivers the same output (originating from a designated party) to a subset of parties. We note that in circuit terminology this can be described as $C_{j,i}$ producing an output associated with sets of parties*.*

A non-interactive h-oracle-aided protocol is one that consists only of a single round of oracle call, and no other communication between the parties.

Consistently with the above formal description, we often refer to M_i as an *interactive circuit* that sends and receives messages (and maintains a state throughout the execution), until finally producing an output after the r-th round of communication.

5.1.4 Correctness and Security of Protocols

Security of multi-party computations is analyzed via the real vs. ideal paradigm. The real model captures the information that can be made accessible to the adversary in an actual execution of the protocol, which includes an arbitrary function of the view of the corrupted parties, as well as honest parties' input and output (but not their internal state during the execution). The ideal model considers a case where the target functionality is computed using oracle access. The protocol is secure if the view of every real adversary can be simulated by an ideal adversary.

We first define the notion of an adversary, note that we slightly deviate from the standard notation and explicitly include the description of the set of corrupted parties as a part of the definition of the adversary. This will be useful for stating our results. We also note that the current definition is syntactic and non-asymptotic and does not address the efficiency of the adversary.

Definition 5.8 (adversaries, the real model, ideal model)**.** *An adversary $(\mathcal{A}, J)$ for an n-party protocol $\Pi = (M_1, \ldots, M_n)$ consists of an interactive circuit $\mathcal{A}$ (sometimes called the adversarial* strategy*), and a set $J \subseteq [n]$. The parties in J (resp. $\overline{J}$) are the* dishonest *(resp.* honest*) parties.*

The execution of Π with input $\mathbf{x}$ under $(\mathcal{A}, J)$ is as follows. The input to $\mathcal{A}$ is the set of inputs $\mathbf{x}[J]$ (the inputs for the parties in J). In each round, $\mathcal{A}$ first receives all messages sent to parties in J from parties in $\overline{J}$, and then outputs messages to be sent to the parties in $\overline{J}$ from the parties in J (i.e. $\mathcal{A}$ is rushing*). At the end of the protocol, $\mathcal{A}$ produces outputs on behalf of all parties in J. The parties in $\overline{J}$ execute according to their respective prescribed M_i algorithms.*

The ordered sequence of outputs of all parties in the execution above is called the output of the real-model execution and denoted as $\mathsf{Real}_{\Pi,J,\mathcal{A}}(\mathbf{x})$. The ideal-model is defined by considering the trivial non-interactive f-oracle-aided protocol Υ^f in which each party simply sends its input x_i to the f-oracle, gets the output y_i from the oracle, and terminates with this output. For an adversary $(\mathcal{A}, J)$ and vector of inputs $\mathbf{x}$, we denote the output of the ideal-model execution by $\mathsf{Ideal}_{f,J,\mathcal{A}}(\mathbf{x})$.

Asymptotic versions. A sequence of functionalities $F = \{f_\kappa\}_{\kappa \in \mathbb{N}}$ is efficiently generated if there exists a PPT algorithm that on input 1^κ outputs a circuit that computes the $n(\kappa)$-party functionality f_κ. A sequence of protocols $\Pi = \{\Pi_\kappa\}$ is efficiently generated if there exists a polynomial time algorithm that takes 1^κ as input and outputs all circuits $C_{j,i}$ associated with Π_κ. A sequence of adversaries $\mathcal{A} = \{\mathcal{A}_\kappa\}$ is (non-uniformly) efficient if there exists a polynomial $p(\cdot)$ such that for every κ the size of the circuit $\mathcal{A}_\kappa$ is at most $p(\kappa)$. We often abbreviate "efficient functionality/protocol/algorithm" and not refer to the sequence explicitly. Throughout this work, we will be concerned with constructing efficiently generated protocols for efficiently generated function ensembles. In fact, our results (implicitly) give rise to a compiler that efficiently converts a finite functionality into a finite protocol.

Definition 5.9 (correctness and security of protocols)**.** *Let $f = \{f_\kappa\}$ be an $n(\kappa)$-party functionality and $\Pi = \{\Pi_\kappa\}$ a (possibly oracle-aided) $n(\kappa)$-party protocol. We say that Π $t(\kappa)$-securely computes f if for every probabilistic non-uniform algorithm $\mathcal{A} = \{\mathcal{A}_\kappa\}$ and every infinite sequence of sets $\{J_\kappa\}$ where $J_\kappa \subseteq [n(\kappa)]$ is of cardinality at most $t(\kappa)$, there exists a probabilistic non-uniform algorithm $\mathcal{B} = \{\mathcal{B}_\kappa\}$ and a polynomial $p(\cdot)$ so that the complexity of $\mathcal{B}_\kappa$ is at most $p(|\mathcal{A}_\kappa|)$, such that for every infinite sequence of inputs $\{\mathbf{x}_\kappa\}$, the distribution ensembles (indexed by κ)*

$$\mathsf{Ideal}_{f_\kappa, J_\kappa, \mathcal{B}_\kappa}(\mathbf{x}_\kappa) \quad \textit{and} \quad \mathsf{Real}_{\Pi_\kappa, J_\kappa, \mathcal{A}_\kappa}(\mathbf{x}_\kappa)$$

are either identical (this is called perfect security), statistically close (this is called statistical security), or computationally indistinguishable (this is called computational security). In the latter case, $\mathcal{A}$ is assumed to be asymptotically efficient.

Note that for an efficiently generated protocol it follows from the definition that the number of parties n, and the input lengths are polynomial in the security parameter κ.

Definition 5.10 (secure reductions, non-interactive reductions)**.** *If there exists a secure h-oracle-aided protocol for computing f, we say that f is reducible to h. If the aforementioned oracle-aided protocol is non-interactive (i.e. only consists of non-adaptive calls to h) we say that the reduction is non-interactive.*

Appropriate composition theorems, e.g., [Gol04, Thms. 7.3.3, 7.4.3], guarantee that the call to h can be replaced by any secure protocol realizing h, without violating the security of the high-level

protocol for f. (In the case of computational security the reduction is required, of course, to be efficient.)

5.1.5 Circuit Representation of a Protocol

Recall that a protocol $\Pi = (M_1, \ldots, M_n)$, is a sequence of interactive circuits. It will be convenient to collapse all these circuits to a single "circuit representation" of a protocol. Informally, we consider the computation of all parties throughout the protocol as parts of one large computation. Each wire of the new circuit is associated with an index corresponding to the party in the protocol that computes this value. This includes the local computations performed by parties throughout the protocol, which are represented as gates whose inputs and outputs are associated with the party who is performing the local computation, and also message transmissions between parties, that are modeled as gates that simply copy their input to the output, where the inputs are associated with the sender and outputs are associated with the receiver. (In this context, we may assume without loss of generality that the circuit employs only transmission gates and there is no need to employ generalized transmission gates.)

Our definition only considers circuits corresponding to *deterministic* protocols. This is both for the sake of simplicity (since we can always consider parties' randomness as a part of their input) and since we will only apply this definition to deterministic protocols in our results.

Definition 5.11 (Circuit Representation of a Protocol)**.** *The* circuit representation *of a deterministic n-party protocol Π is a pair (C, P), where C is a Boolean circuit of size m as defined in Section 5.1.1, and $P : [m] \to [n]$ is a mapping from the wires in C to the n parties. Given a protocol $\Pi = (M_1, \ldots, M_n)$, the circuit C and the mapping P are defined as follows.*

1. *Recalling Definition 5.7, Π consists of a sequence of circuits $C_{j,i}$ which represent the local computation of party i before the j-th round of communication (and a final circuit $C_{r+1,i}$ for the local computation after the last round of communication), we call this the j-th computational step of the protocol.*

2. *All input gates of sub-circuits that correspond to the first step in the protocol are defined as input gates of C. All output wires of sub-circuits that correspond to the last step in the protocol are defined as outputs wires of C.*

3. *The input wires representing the input state of $C_{j,i}$ are connected to the wires representing the output state of $C_{j-1,i}$ via a unary transmission gate. That is, the state of party i in the beginning of computation step j is identical to its state in the end of computation step $j-1$.*

4. *If party i expects a message in step j from party i', then the respective output wire of $C_{j-1,i'}$ is connected to the respective input wire of $C_{j,i}$ via a unary transmission gate. If party i_0 was supposed to send some value via broadcast to multiple parties $i_1, \ldots, i_p$ then a p-ary transmission gates connects the respective output wire in C_{j-1,i_0} to the respective input wires in $C_{j,i_1}, \ldots, C_{j,i_p}$.*

5. *Note that by the description above, the set of wires in C is exactly the union of wires of all circuits $C_{j,i}$. The mapping P associates with party i the wires of circuits $C_{j,i}$, for all j.*

We note that this description implies that for any local gate, all inputs and outputs have the same association. We say that a gate g belongs *to party i if all incoming wires are associated with i. This*

means that all gates in $C_{j,i}$ belong to i, and that the transmission gates that correspond to output wires of $C_{j,i}$ belong to i.

5.2 Key-selection Gadgets

In this section we present two degree-2 randomized gadgets that will be useful for our garbled circuits. Both gadgets take as input one or more selector inputs (scalars from the field) and a pair of keys (vectors over the field), the gadgets release only one of the keys (and some of the selector scalars) depending on the value of the selectors. The notion of "releasing" information will be formalized via the mechanism of perfect randomized encoding of functions (see Section 5.1.2 for formal definition of randomized encoding of functions).

The key-selection function g_{select} takes as input a field element $\gamma \in \mathbb{F}$ referred to as *selector*, and a pair of keys $s^0, s^1 \in \mathbb{F}^\omega$ labeled by "zero" and "one", respectively. The function outputs γ together with $s^{\gamma'}$ where γ', referred to as the *effective selector*, is taken to be γ if γ equals to zero or one, and otherwise, γ' is taken to be 0. Formally, the function is implicitly parameterized by the key-length parameter ω and by a finite field $\mathbb{F}$. (While the gadgets will be used only over fields of characteristic two, all the statements in this subsection hold over an arbitrary finite field.)

Lemma 5.12 (key-selection gadget)**.** *The function $g_{\mathsf{select}}(\gamma, s^0, s^1)$ admits a degree-2 encoding $\mathsf{select}(\gamma, s^0, s^1; R)$ with randomness complexity of 4ω, output complexity of $(1 + 5\omega)$, and computational complexity $O(\omega)$.*

Our second gadget encodes the following, slightly more complicated triple-selection function, g_{tselect}, that will be employed in transmission gates. (The name $\mathsf{tselect}$ stands both for triple-selection and for transmission gate.) The input now consists of three sectors $\gamma_1, \gamma_2, \gamma_3 \in \mathbb{F}$ and again a pair of keys $(s^0, s^1) \in \mathbb{F}^\omega \times \mathbb{F}^\omega$, labeled by zero and one. Let us define a dummy input $\gamma_4 = 0$. The function finds the first binary $\gamma_i \in \{0, 1\}$, denoted as the *effective* selector, and outputs $\gamma_1, \ldots, \gamma_i$ together with the corresponding key s^{γ_i}. In more detail, the output is computed according to the following short program:

- output γ_1.
- If $\gamma_1 \in \{0, 1\}$ output the corresponding key s^{γ_1} and terminate.
- Append to the output the value γ_2 and if $\gamma_2 \in \{0, 1\}$ output s^{γ_2} and terminate.
- Append to the output the value γ_3 and if $\gamma_3 \in \{0, 1\}$ output s^{γ_3} otherwise output s^0.

Lemma 5.13 (triple-selection gadget)**.** *The function $g_{\mathsf{tselect}}(\gamma_1, \gamma_2, \gamma_3, s^0, s^1)$ admits a degree-2 encoding $\mathsf{tselect}(\gamma_1, \gamma_2, \gamma_3, s^0, s^1; R)$ with computational complexity $O(\omega)$. Specifically, the randomness complexity is $74\omega + 15$ and the output complexity is $75\omega + 18$.*

The constants in the lemma were not optimized. In the rest of the section we prove the above lemmas based on several simpler gadgets.

5.2.1 Binary Gadgets

We begin by presenting degree-2 randomized encodings for the functions g_{binSel} and g_{ifnotBin}. The function g_{binSel} gets as an input a scalar $\gamma \in \mathbb{F}$ and two secret vectors, $s^0, s^1 \in \mathbb{F}^\ell$, and outputs

(γ, s^γ) if $\gamma \in \{0,1\}$, and $(\gamma, \perp)$ otherwise. The function g_{ifnotBin} gets as an input a scalar $\gamma \in \mathbb{F}$ and a secret vector $\delta \in \mathbb{F}^\ell$, and outputs (γ, δ) if $\gamma \in \mathbb{F} \setminus \{0,1\}$ and $(\gamma, \perp)$ otherwise. The following encoding are closely related to the ones presented in Section 2.3.

Claim 5.14 (Binary gadgets)**.** *For a length parameter ℓ, let $R_1, R_2 \in \mathbb{F}^\ell$ and define the degree-2 randomized functions*

$$\mathsf{binSel}(\gamma, s^0, s^1; R_1, R_2) := \big(\gamma, \gamma \cdot R_1 + s^0, (1-\gamma) \cdot R_2 + s^1\big),$$
$$\mathsf{ifnotBin}(\gamma, \delta; R_1, R_2) := \big(\gamma, \gamma \cdot R_1, (1-\gamma) \cdot R_2, R_1 + R_2 + \delta\big).$$

Then, binSel *encodes* g_{binSel} *and* $\mathsf{ifnotBin}$ *encodes* g_{ifnotBin} *with arithmetic complexity of $O(\ell)$. Specifically, the output complexity of* binSel *is $2\ell + 1$, the output complexity of* $\mathsf{ifnotBin}$ *is $3\ell + 1$, and, in both cases, the randomness complexity is 2ℓ.*

Proof. For each gadget we present a decoder and simulator. It is not hard to verify that the decoders are perfectly correct, and that the simulators imply the perfect privacy.

The binSel gadget. The decoder receives a triple (γ, a, b), where $\gamma \in \mathbb{F}$ and $a, b \in \mathbb{F}^\ell$. If $\gamma = 0$ then the decoder outputs (γ, a). If $\gamma = 1$ then the decoder outputs (γ, b). If $\gamma \notin \{0,1\}$ the decoder outputs $(\gamma, \perp)$.

The simulator receives a pair (γ, δ), where either (1) $\gamma \in \{0,1\}$ and $\delta \in \mathbb{F}^\ell$, or (2) $\gamma \in \mathbb{F} \setminus \{0,1\}$ and $\delta = \perp$. In case (1), if $\gamma = 0$ the simulator outputs $\mathsf{binSel}(\gamma, \delta, \vec{0}; R_1, R_2)$, and if $\gamma = 1$ the simulator outputs $\mathsf{binSel}(\gamma, \vec{0}, \delta; R_1, R_2)$ where R_1 and R_2 are sampled from $\mathbb{F}^\ell$ and $\vec{0}$ is the all-zero vector of length ℓ. In case (2) the simulator outputs $\mathsf{binSel}(\gamma, \vec{0}, \vec{0}; R_1, R_2)$, where R_1 and R_2 are sampled from $\mathbb{F}^\ell$ and $\vec{0}$ is the all-zero vector of length ℓ.

The $\mathsf{ifnotBin}$ gadget. The decoder receives a tuple (γ, a, b, c), where $\gamma \in \mathbb{F}$ and $a, b, c \in \mathbb{F}^\ell$. If $\gamma \in \{0,1\}$ then the decoder outputs $(\gamma, \perp)$. Otherwise, if $\gamma \notin \{0,1\}$, the decoder outputs $(\gamma, c - \gamma^{-1}a - (1-\gamma)^{-1}b)$.

The simulator receives a pair (γ, δ), where either (1) $\gamma \in \mathbb{F} \setminus \{0,1\}$ and $\delta \in \mathbb{F}^\ell$, or (2) $\gamma \in \{0,1\}$ and $\delta = \perp$. In case (1) the simulator outputs $\mathsf{ifnotBin}(\gamma, \delta; R_1, R_2)$ and in case (2) the simulator outputs $\mathsf{ifnotBin}(\gamma, \vec{0}; R_1, R_2)$, where R_1, R_2 are sampled from $\mathbb{F}^\ell$ and $\vec{0}$ is the all-zero vector of length ℓ. □

5.2.2 Generalized-selection Gadget

Our next building block is a slightly more generalized version of the key-selection gadget. The function g_{gselect} ("g" stands for *generalized*) takes a selector $\gamma \in \mathbb{F}$, pair of zero/one keys, $s^0, s^1 \in \mathbb{F}^\omega$ and an additional "arithmetic key", $\delta \in \mathbb{F}^\ell$, of length ℓ. The function g always outputs γ together with a single key: s^γ if γ is binary, and δ otherwise.

Claim 5.15 (generalized key-selection gadget)**.** *For length parameters ω and ℓ, there exists a degree-2 randomized function* $\mathsf{gselect}$ *that perfectly encodes g_{gselect} with output complexity $2\omega + 3\ell + 1$, randomness complexity of $2\omega + 2\ell$, and computational complexity of $O(\omega + \ell)$.*

Proof. Let $R = (R^1, \ldots, R^4)$ where $R^1, R^2 \in \mathbb{F}^\omega$ and $R^3, R^4 \in \mathbb{F}^\ell$ and define the randomized function

$$\mathsf{gselect}(\gamma, s^0, s^1, \delta; R) := \Big(\mathsf{binSel}(\gamma, s^0, s^1; R^1, R^2), \mathsf{ifnotBin}(\gamma, \delta; R^3, R^4)\Big).$$

Since, by Claim 5.14, binSel and ifnotBin are of degree-2, then gselect is also of degree-2. By the concatenation property of randomized encoding (Lemma 5.4) and Claim 5.14, gselect perfectly encodes the (deterministic) function

$$\mathsf{gselect}'(\gamma, s^0, s^1, \delta) := \Big(g_{\mathsf{binSel}}(\gamma, s^0, s^1), g_{\mathsf{ifnotBin}}(\gamma, \delta)\Big).$$

By composition (Lemma 5.3) it suffices to show that the function gselect′ encodes g_{gselect}.

Correctness is established via the following decoder: (1) Retrieve γ from the output of g_{binSel}. (2) If γ is binary, retrieve s^γ from g_{binSel}, and output the result. (3) Otherwise, retrieve δ from the output of g_{ifnotBin} and output the rseult. It is not hard to verify that the decoder is perfectly correct.

Since the function gselect′ is deterministic, all we need to do in order to prove the privacy is to show how to compute $g_{\mathsf{gselect}}(\gamma, s^0, s^1, \delta)$ given $\mathsf{gselect}'(\gamma, s^0, s^1, \delta)$. The simulator takes γ and a key K, and outputs $((\gamma, K), (\gamma, \perp))$ if $\gamma \in \{0, 1\}$ and $((\gamma, \perp), (\gamma, K))$ otherwise. It is not hard to verify that the simulator computes $\mathsf{gselect}'(\gamma, s^0, s^1, \delta)$.

The output length of the encoding is $2\omega + 3\ell + 2$. We reduce it to $2\omega + 3\ell + 1$, by removing γ from the output of ifnotBin gadget. Since γ is also outputted by binSel this modified encoding still encodes g_{gselect}. (Formally, this optimized version deterministically encodes the un-modified version, and so the claim follow by composition.) □

5.2.3 Proofs of Lemma 5.13 and Lemma 5.12

We can now prove Lemmas 5.12 and 5.13.

Proof of Lemma 5.12. We derive Lemma 5.12 by setting $\delta = s^0$ in the generalized key-selection gadget. That is, we set the parameter ℓ to ω and define $\mathsf{select}(\gamma, s^0, s^1; R) := \mathsf{gselect}(\gamma, s^0, s^1, s^0; R)$. By the substitution property (Lemma 5.5), the resulting function encodes g_{select}. The complexity statements follow from the complexity of gselect. Moreover, since gselect is of degree-2, so is select, and Lemma 5.12 follows. □

Proof of Lemma 5.13. The encoding uses randomness R that will be parsed into 5 vectors $R = (K_1, K_2, R_1, R_2, R_3)$ whose length will be defined later. The encoding $\mathsf{tselect}(\gamma_1, \gamma_2, \gamma_3, s^0, s^1, \delta; R)$ is defined by

$$\Big(\mathsf{gselect}(\gamma_1, s^0, s^1, K_1; R_1), K_1 + \mathsf{gselect}(\gamma_2, s^0, s^1, K_2; R_2), K_2 + \mathsf{select}(\gamma_3, s^0, s^1; R_3)\Big).$$

Let us start by analyzing the output complexity and randomness complexity starting from the last entry and going backward to the first one. The randomness R_3 for the last select gadget is of length 4ω, and, since the output length of this gadget is $\ell = 5\omega+1$, the length of K_2 (which blinds it) is also ℓ. The second gselect gadget is therefore employed with length parameters ω and ℓ and therefore it employs $2\omega + 2\ell$ random elements, i.e., $R_2 \in \mathbb{F}^{2\omega+2\ell}$, and its output length is $\ell' = 2\omega + 3\ell + 1$. Consequently, K_1 is of length ℓ' and so the first gselect gadget is employed with length parameters ω and ℓ'. This means that the randomness R_1 is of length $2\omega + 2\ell'$ and the output length of the first entry is $2\omega + 3\ell' + 1$. Overall the output length is $(2\omega + 3\ell' + 1) + \ell' + \ell = 75\omega + 18$ and the randomness complexity is $(2\omega + 2\ell') + \ell' + (2\omega + 2\ell) + \ell + 4\omega = 74\omega + 15$. Since gselect and select are of degree-2, so is tselect.

We prove that $\mathsf{tselect}$ perfectly encodes g_{tselect}. By the concatenation property of randomized encoding (Lemma 5.4), and by Claim 5.14 and Lemma 5.12, $\mathsf{tselect}$ perfectly encodes the (deterministic) function $\mathsf{tselect}'(\gamma_1, \gamma_2, \gamma_3, s^0, s^1, \delta, K_1, K_2)$ defined via

$$\Big(g_{\mathsf{gselect}}(\gamma_1, s^0, s^1, K_1), K_1 + g_{\mathsf{gselect}}(\gamma_2, s^0, s^1, K_2), K_2 + g_{\mathsf{select}}(\gamma_3, s^0, s^1)\Big).$$

Note that K_1 is longer than $g_{\mathsf{gselect}}(\gamma_2, s^0, s^1, K_2)$ (and similarly K_2 is longer than $g_{\mathsf{select}}(\gamma_3, s^0, s^1)$). We resolve this syntactic mismatch via concatenation of zeroes. That is, we use the convention that whenever an L-long vector v is added to a vector u whose length is only $L - k$, the sum is defined to be $v + (u \circ 0^k)$.

By the composition lemma (Lemma 5.3) it suffices to show that $\mathsf{tselect}'$, viewed as a randomized function with randomness K_1, K_2, perfectly encodes the function g_{tselect}. We begin by correctness. The decoder retrieves the value γ_1 and a key K from the first output (of g_{gselect}). If γ_1 is binary, the decoder outputs (γ_1, K), where K, by the definition of g_{gselect} will indeed be s^{γ_1}. Otherwise (if γ_1 is non-binary), the decoder learns $K = K_1$ and uses it to un-pad the second entry. The decoder now holds the pair $(\gamma_2, K') = g_{\mathsf{gselect}}(\gamma_2, s^0, s^1, K_2)$. If γ_2 is binary, the decoder outputs (γ_1, γ_2, K'), where K', by the definition of g_{gselect}, will be s^{γ_2}. Otherwise, (if γ_2 is non-binary) the decoder holds $K' = K_2$ and can use it to un-pad the last entry $(\gamma_3, s') = g_{\mathsf{select}}(\gamma_3, s^0, s^1)$ where s' equals to s^1 if $\gamma_3 = 1$, and equals to s^0 otherwise. The decoder outputs $(\gamma_1, \gamma_2, \gamma_3, s')$. This completes the description of the decoder. Perfect correctness follows directly.

To prove perfect privacy, we describe a simulator. The input to the simulator can be either (1) (a, s) where $a \in \{0, 1\}$ and $s \in \mathbb{F}^\omega$, or (2) (a, b, s) where $a \in \mathbb{F} \setminus \{0, 1\}$, $b \in \{0, 1\}$ and $s \in \mathbb{F}^\omega$, or (3) (a, b, c, s) where $a, b \in \mathbb{F} \setminus \{0, 1\}$, $c \in \mathbb{F}$ and $s \in \mathbb{F}^\omega$. The simulator samples keys $K_1 \in \mathbb{F}^{\ell'}$ and $K_2 \in \mathbb{F}^\ell$ and does the following: In case (1), the simulator outputs the tuple $((a, s), K_1, K_2)$, in case (2) the simulator outputs the tuple $((a, K_1), K_1 + (b, s), K_2)$, and in case (3) the simulator outputs the tuple $((a, K_1), K_1 + (b, K_2), K_2 + (c, s))$. It is not hard to verify that the simulator perfectly samples the desired distribution. The lemma follows. □

5.3 The Garbled Circuit

In this section, we describe and analyze a new garbling technique for Boolean circuits that works natively over a binary extension field $\mathbb{F}$. The garbled circuit scheme employs the *key-selection* and *triple-selection* gadgets that appear in Section 5.2, and it consists of three algorithms: pre, enc and dec. All three algorithm take as an input a Boolean circuit C, consisting of n_{inp} input gates, n_{cmp} local computation gates and n_{trns} transmission gates, with m wires in total, together with several additional values. When the circuit is clear from the context we omit it from the input. Otherwise, we use the notation enc^C and pre^C to denote that C is the input circuit to enc and pre. We further note that enc and dec depend only the topology of C. The following subsections are devoted to the descriptions of each of these subroutines. The analysis appears in Section 5.3.2 and some of the proofs are deferred to Appendix C.

5.3.1 Description of the Garbled Circuit

In the following section we describe the three algorithms $\mathsf{pre}, \mathsf{enc}$ and dec. We begin with the description of the preprocessing algorithm pre.

Algorithm pre

Inputs: The function pre is parameterized by the circuit C, and, in addition, pre receives an input vector $\mathbf{x} \in \mathbb{F}^{n_{\mathsf{inp}}}$ for the circuit C, and a vector of wire-masks $\boldsymbol{\alpha} = (\alpha_j)_{j\in[m]} \in \mathbb{F}^m$ that consists of a mask $\alpha_j \in \mathbb{F}$ for each wire $j \in [m]$.

Output: The output of $\mathsf{pre}^C(\mathbf{x}, \boldsymbol{\alpha})$ is a vector of $n_{\mathsf{inp}} + 4n_{\mathsf{cmp}} + m$ field elements.

Computation: For every input gate g in C, with input-value x_g, define

$$\boldsymbol{\Gamma}_g := x_g.$$

For every local computation gate g in C, computing a function $G : \{0,1\}^2 \to \{0,1\}$, with incoming wires $c, d \in [m]$, outgoing wire $k \in [m]$, define

$$\boldsymbol{\Gamma}_g = (\gamma_g^{0,0}, \gamma_g^{0,1}, \gamma_g^{1,0}, \gamma_g^{1,1}) \in \mathbb{F}^4 \quad \text{where } \gamma_g^{\beta_c,\beta_d} := G(\beta_c - \alpha_c, \beta_d - \alpha_d), \quad \forall \beta_c, \beta_d \in \{0,1\}.$$

The output of pre is $\boldsymbol{\Gamma} := (\boldsymbol{\Gamma}_g)_{g\in C} \circ \boldsymbol{\alpha}$, the concatenation of all $\boldsymbol{\Gamma}_g$ together with the vector $\boldsymbol{\alpha}$ of wire masks.

Figure 12: Algorithm pre

Observe that pre^C can be computed by making $O(m)$ arithmetic operations.

Remark 5.16 (locality of pre)**.** *It is important to note that* pre *is a* local function*. That is, when g is an input gate, $\boldsymbol{\Gamma}_g$ depends only on the corresponding input, and when g is a local computation gate, $\boldsymbol{\Gamma}_g$ depends only on the wire-masks of the incoming wires. Moreover, $\boldsymbol{\Gamma}_g$ only depends on the circuit C locally, i.e., it can be computed based on the description of the gate g which consists of the indices of the incoming wires and the Boolean operator that the gate computes. It is also useful to keep in mind that when these masks are binary, then so are the entries of $\boldsymbol{\Gamma}_g$.*

Encoding function. The input to the enc function (in addition to the circuit C) is a vector of $n_{\mathsf{inp}}+4n_{\mathsf{cmp}}+m$ field elements that we parse into $(\boldsymbol{\Gamma}_g)_{g\in C}$ and $(\alpha_j)_{j\in[m]}$ according the output format of pre. The enc function will also employ random field elements (that should be viewed as internal randomness). As in "standard" garbled circuits, this randomness includes a vector of wire-keys $\mathbf{s} = (s_j^0, s_j^1)_{j\in[m]}$ that contains, for each wire $j \in [m]$, a pair of keys (s_j^0, s_j^1) of length ω_j each. Each of these keys s_j^b is partitioned into two halves, denoted by $s_j^b[0]$ and $s_j^b[1]$. The key-length ω_j will be defined later in a recursive manner starting from the output wires whose keys will be the empty strings (i.e., $\omega_j = 0$ for an output wire j). We will also employ additional randomness that will be used as internal randomness for the key-selection and the triple-selection gadget. Semantically, the output of enc should allow us to learn, for each wire j, a masked value $v_j + \alpha_j$, where v_j denotes the value of the j-th wire under the input $\mathbf{x}$ (implicitly given as part of $\boldsymbol{\Gamma}$ vector), together with the wire's key $s^{v_j+\alpha_j}$. Formally, the randomized function $\mathsf{enc}^C(\boldsymbol{\Gamma})$ creates a gate table Q_g for every non-output gate g and outputs $\mathbf{Q} = (Q_g)_{g\in C}$. The gate tables are computed differently for input gates, computation gates, and transmission gates according to the following subroutines.

Algorithm enc

Inputs: The function enc is parameterized by the circuit C, and, in addition, enc receives an input vector of $n_{\mathsf{inp}} + 4n_{\mathsf{cmp}} + m$ field elements, parsed as $\boldsymbol{\Gamma} = (\boldsymbol{\Gamma}_g)_{g\in C}$ and $\boldsymbol{\alpha} = (\alpha_j)_{j\in[m]}$.

Randomness: A vector of wire keys $\mathbf{s} = (s_j^0, s_j^1)_{j\in[m]}$ and a vector of randomness $\mathbf{R}$ to the key-selection and triple-selection gadgets.

Output: The ouptut of $\mathsf{enc}^C(\mathbf{\Gamma}, \boldsymbol{\alpha}; \mathbf{s}, \mathbf{R})$ is a sequence $(Q_g)_{g\in C}$ of gate tables for every non-output gate in C.

Computation: For a (non-output) gate g the table Q_g is defined as follows.

- *Input gate.* Let x_g denote the input-gate value (extracted from $\mathbf{\Gamma}_g$), let k denote the index of the outgoing wire, and let α_k denote the corresponding mask. Our goal is to view $x_g + \alpha_k$ as a public selector and release the value $s_k^{x_g+\alpha_k}$ if the selector is binary, and to release s_k^0 otherwise. We therefore define the gate table Q_g as:

$$Q_g := \mathsf{select}(x_g + \alpha_k, s_k^0, s_k^1; \mathbf{R}_g),$$

  where $\mathbf{R}_g$ is random vector of length $5\omega_k$ (since the key length of s_k^0, s_k^1 is ω_k). The total length of Q_g is therefore $5\omega_k + 1$.

- *Local gate.* For every local computation gate g in C, with incoming wires c, d and outgoing wire k, let $(\gamma_g^{0,0}, \gamma_g^{0,1}, \gamma_g^{1,0}, \gamma_g^{1,1})$ and $(\alpha_c, \alpha_d, \alpha_k)$ be the corresponding gate-values and wire-masks from $\mathbf{\Gamma}$. Semantically, our goal is to map the keys of the incoming wires $s_c^{\beta_c}$ and $s_d^{\beta_d}$, for any given $\beta_c, \beta_d \in \{0,1\}$, to the corresponding key $s_k^{\gamma_g^{\beta_c,\beta_d}+\alpha_k}$ of the outgoing wire. We will make sure that this will be the case only when the masked value $\gamma_g^{\beta_c,\beta_d} + \alpha_k$ is binary, and otherwise will release the zero-key s_k^0. Formally, for every $\beta_c, \beta_d \in \{0,1\}$, we define

$$q_g^{\beta_c,\beta_d} := \mathsf{select}(\gamma_g^{\beta_c,\beta_d} + \alpha_k, s_k^0, s_k^1; \mathbf{R}_{g,\beta_c,\beta_d}), \quad \text{where } \mathbf{R}_{g,\beta_c,\beta_d} \leftarrow \mathbb{F}^{5\omega_k},$$

  and construct the entry $Q_g^{\beta_c,\beta_d}$ as a one-time pad encryption of $q_g^{\beta_c,\beta_d}$ with key $s_c^{\beta_c}[\beta_d] + s_d^{\beta_d}[\beta_c]$:

$$Q_g^{\beta_c,\beta_d} := q_g^{\beta_c,\beta_d} + \left(s_c^{\beta_c}[\beta_d] + s_d^{\beta_d}[\beta_c]\right).$$

  Finally, we define $Q_g := (Q_g^{0,0}, Q_g^{0,1}, Q_g^{1,0}, Q_g^{1,1})$. The length of $q_g^{\beta_c,\beta_d}$ is $6\omega_k + 1$, and so the required length for each half of s_c and s_d, is $5\omega_k + 1$. Accordingly, the full length of these keys, $\omega_c = |s_c|$ and $\omega_d = |s_d|$, is $2(5\omega_k + 1)$.

- *Generalized transmission gate.* For every transmission gate g with incoming wire c, outgoing wires $k_1, \ldots, k_p$, that are labeled by $(a_j, b_j)_{j\in[p]}$, the table Q_g, of g consists of two parts (Q_g^0, Q_g^1), each of which further constitutes of p sub-parts. For every $\beta_c \in \{0,1\}$, we define an entry $Q_g^{\beta_c}$ as follows. (See intuition in Remark 5.17 below.) Let $\delta_c := \beta_c - \alpha_c$ and $\delta_j := a_j \cdot \delta_c + b_j$ and let

$$\begin{aligned} q_g^{\beta_c}[j] &:= \mathsf{tselect}(\delta_j + \alpha_{k_j}, \alpha_{k_j}, \delta_j, s_{k_j}^0, s_{k_j}^1; \mathbf{R}_g^j), \\ q_g^{\beta_c} &:= (q_g^{\beta_c}[1], \ldots, q_g^{\beta_c}[p]) \\ Q_g^{\beta_c} &:= q_g^{\beta_c} + s_c^{\beta_c}. \end{aligned}$$

  The entries of the vector $\mathbf{R}_g^j$ are chosen uniformly from $\mathbb{F}$ and its length is defined by the specification of the $\mathsf{tselect}$ gadget. Recall that the randmness/output complexity of the gadget is $K \cdot \omega_{k_j} + K'$ for some constants K, K'. It follows that the length ω_c, of both s_c^0 and s_c^1, is $|q_g^{\beta_c}| \le \sum_{i\in[p]} K \cdot \omega_{k_i} + K' \le K \cdot (\max_{i\in[p]} \omega_{k_i}) + K'$.

  We emphasize that the labels $(a_j, b_j)_{j\in[p]}$ are fixed constants that are given as part of the description of the circuit C. Therefore, δ_j is a linear function of the input α_c and the expression Q_g is a degree-2 function in the α's and the internal randomness.

The output of enc is the sequence $(Q_g)_{g\in C}$ of gate tables for every non-output gate in C.

Figure 13: Algorithm enc

Remark 5.17 (Generalized transmission gate: intuition.)**.** *Let us briefly explain the construction of transmission gates. For simplicity, we assume that $a_j = 1$ and $b_j = 0$ for all $j \in [p]$ (this is the case where g is a simple transmission gate).*

For every $\beta_c \in \{0,1\}$ the ciphertext $Q_g^{\beta_c}$ should allow us to map the key $s_c^{\beta_c}$ of the incoming wire to a corresponding key $s_{k_j}^{\beta_c-\alpha_c+\alpha_{k_j}}$ of the j-th output wire. Recall that $\delta_c = \delta_j = \beta_c - \alpha_c$, and so, under this notation, we should encrypt the outgoing key $s_{k_j}^{\delta_j+\alpha_{k_j}}$ under the incoming key $s_c^{\beta_c}$. Note that $Q_g^{\beta_c}$ encrypts $q_g^{\beta_c}$ under the incoming key $s_c^{\beta_c}$, and so the entry $q_g^{\beta_c}[j]$ should release the key $s_{k_j}^{\delta_j+\alpha_{k_j}}$ if the masks are all binary, and should release some default values otherwise. Specifically, the definition of $q_g^{\beta_c}[j]$ should take into account the case where either the mask α_c of the input wire or the mask α_{k_j} of the outgoing wire are non-binary. In the former case, we would like to effectively fix α_c to zero (or to any other canonical value that is known to the adversary), and in the latter case to fix α_{k_j} to zero. Things get complicated since these values should be hidden and so we cannot use them as public-selectors to our gadgets. Instead, we apply our gadget on the masked values in a way that guarantees essentially the same result.

Recall that $\mathbb{F}$ is an extension field of the binary field, and therefore the value δ_c is binary if and only if α_c is binary. Also, $\delta_j + \alpha_{k_j}$ is binary if and only if either (a) both, α_c and α_{k_j}, are binary or (b) both values are non-binary, but their sum is binary. In the latter case both wires are essentially controlled by the adversary, and so this scenario is less interesting and will be ignored for now. Let us now take a close look at $q_g^{\beta_c}[j]$. If $\delta_j + \alpha_{k_j} \in \{0,1\}$ (essentially both masks are binary as per option (a) above) the wire-key $s_{k_j}^{\delta_j+\alpha_{k_j}}$ can be extracted. On the other hand, if $\delta_j + \alpha_{k_j} \notin \{0,1\}$, i.e., at least one of the masks is non-binary, then the tselect *gadget moves on to the selector α_{k_j}. If, in addition, $\alpha_{k_j} \in \{0,1\}$ (i.e., the outgoing wire-mask is binary) then the wire-key $s_{k_j}^{\alpha_{k_j}}$ is being released, effectively setting the* real value *of the outgoing wire to zero. Jumping a head, this is fine since the adversary controls the input to the transmission gate anyway and so he can set the output to zero. (Also note that α_{k_j} is being leaked by the gadget but again this will be fine since the real value of this wire is known to the adversary anyway.) Otherwise, (i.e., the mask of the outgoing wire is non-binary) the gadget proceeds to the third selector δ_j. If δ_j is binary it releases the wire-key $s_{k_j}^{\delta_j}$ (effectively setting the mask of the outgoing wire to zero), and if δ_j is also non-binary then $s_{k_j}^0$ is being released.*

Complexity of enc**.** The arithmetic complexity of computing the table of a gate g is linear in the length ω_j of the longest wire-key j that is adjacent to g. (For input gate j is the outgoing wire, and for other gates j can be taken to be any of the two incoming wires.) The length of each such key is at most exponential in the depth of the circuit. (Recall that for depth computation, the cost of a p-transmission gate is $\log p$). Hence, the total arithmetic complexity is $\mathrm{poly}(m, 2^d)$. Consequently the bit-length of the output and randomness is $\mathrm{poly}(m, 2^d, \log|\mathbb{F}|)$. Furthermore, each gate table is a degree-2 function in the input and the randomness.

Decoding procedure. The decoding procedure dec takes as input the circuit C (or actually its topology) and a sequence of gate tables $\mathbf{Q}$, and outputs a binary vector $(v_j)_{j\in[m]}$.

Algorithm dec

Inputs: The function dec is parameterized by the circuit C, and, in addition, dec receives a sequence of gate tables $\mathbf{Q} = (Q_g)_{g\in C}$.

Output: The output of $\mathsf{dec}(\mathbf{Q})$ is a binary vector $(v_j)_{j\in[m]}$.

Computation: The computation is done by traversing the gates of the circuit from the inputs to the outputs in topological order, and for each wire j computing a pair $(v_j, s_j^{v_j})$, where $s_j^{v_j} \in \mathbb{F}^{\omega_j}$, as follows.

- *Input gate.* For a wire k coming out of an input gate g, apply the decoder of the select gadget to Q_g, extract from the output the effective selector $\gamma \in \{0,1\}$ and the selected key s and set $v_k = \gamma$ and $s_k^{v_k} = s$. (Recall that the effective selector can be recovered given the output of the g_{select}-decoder.)
- *Local gate.* For a wire k that is an output wire of a local gate g with incoming wires c and d, for which the values $(v_c, s_c^{v_c})$ and $(v_d, s_d^{v_d})$ were already computed, do the following. Retrieve $Q_g^{v_c,v_d}$ from the gate table Q_g, and set
$$q_g^{v_c,v_d} := Q_g^{v_c,v_d} - (s_c^{v_c}[v_d] + s_d^{v_d}[v_c]),$$
then apply the decoder of the select gadget to $q_g^{v_c,v_d}$, extract from the output the effective selector $\gamma \in \{0,1\}$ and the selected key s and set $v_k = \gamma$ and $s_k^{v_k} = s$.
- *Generalized Transmission gate.* For wires $k_1, \ldots, k_p$ that are outputs of a transmission gate g with incoming wire c, for which the pair $(v_c, s_c^{v_c})$ was already computed, we do the following. Retrieve $Q_g^{v_c}$ from the gate table Q_g, and set
$$q_g^{v_c} := Q_g^{v_c} - s_c^{v_c},$$
parse $q_g^{v_c}$ as $(q_g^{v_c}[1], \ldots, q_g^{v_c}[p])$ and, for each $i \in [p]$, apply the decoder of the $\mathsf{tselect}$ gadget to $q_g^{v_c}[i]$, extract from the output the effective selector $\gamma \in \{0,1\}$ and the selected key s and set $v_{k_i} = \gamma$ and $s_{k_i}^{v_{k_i}} = s$. (Recall that the effective selector can be recovered given the output of the g_{tselect}-decoder.)

The output of dec is the binary vector $(v_j)_{j\in[m]}$.

Figure 14: Algorithm dec

The total arithmetic complexity of dec is linear in $\sum_j \omega_j$, which is $\mathrm{poly}(m, 2^d)$, as we showed in the analysis of enc.

5.3.2 Analysis of the Garbled Circuit

Let us start by summarizing the syntactic properties of $\mathsf{pre}, \mathsf{enc}$ and dec.

Proposition 5.18. *For a circuit C of size m and depth d, the arithmetic complexity of* $\mathsf{pre}, \mathsf{enc}, \mathsf{dec}$ *is* $\mathrm{poly}(m, 2^d)$. *In addition, the randomized function* enc *is of degree 2.*

The standard security of information-theoretic garbled circuit [IK02] essentially says that for binary inputs $\mathbf{x}$, the function $C(\mathbf{x})$ is perfectly encoded by the randomized function $C'(\mathbf{x})$ which samples uniform wire masks $\boldsymbol{\alpha} \in \{0,1\}^m$, and outputs $\mathsf{enc}(\mathsf{pre}(\mathbf{x}, \boldsymbol{\alpha}))$ together with the masks of the output wires. In fact, as implicitly observed in [ABT18], this can be extended to the case where $\boldsymbol{\alpha} \in \{0,1\}^m$ is treated as part of the input. That is, the randomized function $\mathsf{enc}(\mathsf{pre}(\mathbf{x}, \boldsymbol{\alpha}))$ defined over binary inputs $(\mathbf{x}, \boldsymbol{\alpha})$, perfectly encodes the "garbled evaluation" function $\mathsf{gEval}_C(\mathbf{x}, \boldsymbol{\alpha})$ that

applies C to $\mathbf{x}$, computes, for each wire i, the intermediate value u_i (induced by the input $\mathbf{x}$) and outputs the masked values $(u_i + \alpha_i)_{i\in[m]}$.

Below, we show that even when, for some subset of the gates I, the value of pre is maliciously set to some arbitrary $\mathbf{\Gamma}_I$, the residual function $\mathsf{enc}(\mathbf{\Gamma}_I, \mathsf{pre}(\mathbf{x}[\bar{I}_{\mathsf{inp}}], \boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}]))$ restricted to inputs $\mathbf{x}[\bar{I}_{\mathsf{inp}}]$ of the input gates $\bar{I}_{\mathsf{inp}}$ outside I and masks $\boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}]$ of the wires $\bar{I}_{\mathsf{wire}}$ that do not enter the I-gates, still encodes a related function $\mathsf{gEval}_{\hat{C}}((\hat{\mathbf{x}}[I_{\mathsf{inp}}], \mathbf{x}[\bar{I}_{\mathsf{inp}}]), (\hat{\boldsymbol{\alpha}}[I_{\mathsf{wire}}], \boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}]))$ where $\hat{C}$ is obtained from C by corrupting the gates in I and $\hat{\mathbf{x}}[I_{\mathsf{inp}}]$ and $\hat{\boldsymbol{\alpha}}[I_{\mathsf{wire}}]$ are "synthetic" binary inputs/masks for the I input gates I_{inp} and for the wires I_{wire} that enter I-gates. (The modified circuit and the syntectic values are induced by $\mathbf{\Gamma}_I$.) Such a statement implicitly appears in [ABT19] for a binary version of the construction, and we extend it to the arithmetic setting via the above construction.

In order to formalize the above, we need several definitions and notational conventions.

Notation. The following definitions hold with respect to some fixed circuit C. Fix a set of gates I. The set of wires associated with I, denoted by I_{wire}, consist of all wires that *go into* a gate in I. We denote the set of input wires that are associated with I by I_{inp}, and the set of output wires that are associated with I by I_{out}. An *I-input vector*, denoted $\mathbf{x}[I_{\mathsf{inp}}]$, is a tuple of field elements that contains an element x_g for each input gate g in I. An *I-mask vector*, denoted, $\boldsymbol{\alpha}[I_{\mathsf{wire}}]$, is a tuple of mask elements that contains an element α_i for each wire i in I_{wire}. An *I-gate vector*, $\mathbf{\Gamma}_I$, is a vector of field elements that consists of an I-input vector, an I-mask vector, and a tuple $\mathbf{\Gamma}_g \in \mathbb{F}^4$ for every local computation gate g in I. We let pre_I denote the restriction of pre to I. That is, pre_I takes an I-input vector and an I-mask vector, and outputs the corresponding I-gate vector $\mathbf{\Gamma}_I$. The locality of pre (Remark 5.16), ensures that pre_I is well-defined. We let $\bar{I} = \{g \notin I\}$ denote the complement of I and note that $\bar{I}_{\mathsf{wire}}$ is the complement of I_{wire}.

Admissible set of gates. The set I typically denotes the set of gates that are controlled by the adversary. Such a set must be *admissible* in the following sense. Consider the graph G obtained by cutting the outgoing wires of all (possibly generalized) transmission gates, then I is admissible if it contains a subset of the connected components of the graph. Equivalently, the set I is admissible if for every wire connecting a gate $g \in \bar{I}$ to a gate $g' \in I$, the gate g is a (possibly generalized) transmission gate and the gate g' is a local gate, and, for every wire connecting a gate $g \in I$ to a gate $g' \in \bar{I}$, the gate g is a (possibly generalized) transmission gate and the gate g' is a local gate. (Recall that a transmission gates and generalized transmission gates are always connected to local computation gates.)

Remark 5.19 (Admissible sets.)**.** *The definition of admissible sets is motivated by the fact that in a circuit representation of a protocol (see Definition 5.11), the set of all gates that belong to the adversary is always an admissible set.*

The definition of admissible sets implies that all wires connected to local-gates and input-gates in I are in I_{wire}, and all wires connected to local-gates and input-gates in $\bar{I}$ are in $\bar{I}_{\mathsf{wire}}$. Furthermore, an incoming wire to a transmission gate in I is necessarily in I_{wire} (however, an outgoing wire might belong to $\bar{I}_{\mathsf{wire}}$), and an incoming wire to a transmission gate in $\bar{I}$ is necessarily in $\bar{I}_{\mathsf{wire}}$ (however an outgoing wire might belong to I_{wire}).

I-corrupted circuit. We say that a circuit $\hat{C}$ is an *I-corrupted* version of C if $\hat{C}$ is a Boolean circuit that differs from C only with respect to gates that belong to I. Specifically, the topology

of the circuit remains unchanged (which also means that the input gates remains unchanged). Every local computation gate $g \in I$ may be changed to compute an arbitrary Binary operator $G : \{0,1\} \times \{0,1\} \to \{0,1\}$. Finally, every simple/generalized transmission gate g can be modified to a different generalized transmission gate with the following important restriction: If g is $\bar{I}$-consistent then so is the modified gate g'. Here we say that a transmission gate is $\bar{I}$-consistent if the labels of all outgoing wires that are associated with $\bar{I}$ are the same. Specifically, this means that if g is a broadcast gate then the gate still broadcasts a single value to all the outgoing wires that are not associated with I. (We emphasize that a simple transmission gate g can be replaced by a generalized transmission gate g'.)

For an admissible set of gates I, and an arbitrary I-gate vector $\mathbf{\Gamma}_I$, the following lemma captures the information that is given by the outcome of $\mathsf{enc}(\mathbf{\Gamma}_I, \mathsf{pre}_I(\mathbf{x}[\bar{I}_{\mathsf{inp}}], \boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}]))$ where $\mathbf{x}[\bar{I}_{\mathsf{inp}}]$ is a binary $\bar{I}$-input vector and $\boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}]$ is a binary mask vector.

Lemma 5.20 (Main lemma)**.** *Let I be an admissible set of gates. There exists mapping T, with arithmetic complexity $\mathrm{poly}(m)$, that takes an I-gate vectors $\mathbf{\Gamma}_I$ and outputs a triple $(\hat{\mathbf{x}}[I_{\mathsf{inp}}], \hat{\boldsymbol{\alpha}}[I_{\mathsf{wire}}], \hat{C})$ of a binary I-input vector $\hat{\mathbf{x}}[I_{\mathsf{inp}}]$, a binary I-mask vector $\hat{\boldsymbol{\alpha}}[I_{\mathsf{wire}}]$, and an I-corrupted circuit $\hat{C}$ of C such that the following hold.*

1. *For every fixed $\mathbf{\Gamma}_I$, the deterministic function $g_{\mathbf{\Gamma}_I}$, that given a binary $\bar{I}$-input vector $\mathbf{x}[\bar{I}_{\mathsf{inp}}]$ and a binary $\bar{I}$-mask vector $\boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}]$ outputs the value*

$$\mathsf{gEval}_{\hat{C}}\left((\hat{\mathbf{x}}[I_{\mathsf{inp}}], \mathbf{x}[\bar{I}_{\mathsf{inp}}]), (\hat{\boldsymbol{\alpha}}[I_{\mathsf{wire}}], \boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}])\right), \quad \text{where } (\hat{\mathbf{x}}[I_{\mathsf{inp}}], \hat{\boldsymbol{\alpha}}[I_{\mathsf{wire}}], \hat{C}) = T(\mathbf{\Gamma}_I),$$

   is perfectly encoded by the randomized function $\hat{g}_{\mathbf{\Gamma}_I}(\mathbf{x}[\bar{I}_{\mathsf{inp}}], \boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}])$ that outputs

$$\mathsf{enc}^C\left(\mathbf{\Gamma}_I, \mathsf{pre}_I^C(\mathbf{x}[\bar{I}_{\mathsf{inp}}], \boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}])\right).$$

   The arithmetic complexity of the encoding is $\mathrm{poly}(m, 2^d)$, the decoder is the one presented in Section 5.3 and can be computed in time $\mathrm{poly}(m, 2^d)$ independently of $\mathbf{\Gamma}_I$, and the simulator can be computed with arithmetic complexity of $\mathrm{poly}(m, 2^d)$, given the parameters $(I, \mathbf{\Gamma}_I)$.

2. *Moreover, if $\mathbf{\Gamma}_I$ is honestly generated via $\mathsf{pre}_I^C(\mathbf{x}[I_{\mathsf{inp}}], \boldsymbol{\alpha}[I_{\mathsf{wire}}])$ for some binary I-input vector $\mathbf{x}[I_{\mathsf{inp}}]$ and binary I-mask vector $\boldsymbol{\alpha}[I_{\mathsf{wire}}]$, then $T(\mathbf{\Gamma}_I)$ outputs $(\mathbf{x}[I_{\mathsf{inp}}], \boldsymbol{\alpha}[I_{\mathsf{wire}}], C)$.*

Recall that the garbled-evaluation function $\mathsf{gEval}_{\hat{C}}$ takes a binary input-vector $\mathbf{x}$ and a binary vector of mask values $\boldsymbol{\alpha}$, computes, for each wire i of $\hat{C}$, the intermediate value u_i (induced by the input $\mathbf{x}$) and outputs the masked values $(u_i + \alpha_i)_{i \in [m]}$.

The proof of Lemma 5.20 is deferred to Section C.

5.4 The Master Theorem

Our master theorem closely follows that of [ABT19], with $\mathbb{F}_2$ now replaced with $\mathbb{F}$.

Theorem 5.21 (Master theorem)**.** *For every n-party protocol Π whose circuit representation is of size m and depth d, there exists an n-party non-interactive oracle aided protocol $\hat{\Pi}^h$ over finite field $\mathbb{F}$ of characteristic 2 with the following properties.*

1. ***Efficiency.*** *The communication and computational complexity of $\hat{\Pi}^h$ is at most $L = \mathrm{poly}(2^d, n, m, \log|\mathbb{F}|)$ times larger than that of Π.*

2. ***Quadratic Oracle.*** *The randomized oracle h can be written as a quadratic function over $\mathbb{F}$ in the inputs and in the internal randomness.*[10]

3. ***Simulation.*** *For every strategy $\hat{\mathcal{A}}$ acting on $\hat{\Pi}^h$, there exists a strategy $\mathcal{A}$ of complexity at most $L = \mathrm{poly}(2^d, n, m, \log|\mathbb{F}|)$ times larger acting on Π, such that for all $J \subseteq [n]$ and for all $\mathbf{x} = (x_1, \ldots, x_n)$, the distribution of $\mathsf{Real}_{\Pi,J,\mathcal{A}}(\mathbf{x})$ and $\mathsf{Real}_{\hat{\Pi}^h,J,\hat{\mathcal{A}}}(\mathbf{x})$ are identical. Furthermore, if $\hat{\mathcal{A}}$ is semi-honest (i.e. follows the protocol) then so is $\mathcal{A}$.*

Remark 5.22 (constructiveness)**.** *We mention that Theorem 5.21 is* constructive *in the sense that given a description of Π (and the size of $\mathbb{F}$) one can generate a description of $\hat{\Pi}$ and h in time L (as defined above). Moreover, given the code of $\hat{\mathcal{A}}$ (and the code of Π) can generate the code of $\mathcal{A}$ in time $L\mathrm{poly}(|\hat{\mathcal{A}}|)$. (In fact, $\mathcal{A}$ makes a black-box use of $\hat{\mathcal{A}}$.)*

The protocol $\hat{\Pi}^h$ essentially computes the garbled-circuit encoding (Section 5.3) of the circuit representation of Π (Section 5.1.5). The $\mathbf{\Gamma}$ values are precomputed by the parties and are then delivered to the oracle h that corresponds to the procedure enc. Given the output of enc, each party recovers the garbled evaluation of Π via the decoder dec and then unmasks the values of her own wires. The security is based on Lemma 5.20. Details follow.

Proof. It suffices to prove the theorem only for *deterministic* protocols Π, since for a randomized protocol we can always consider Π to be the induced deterministic protocol where the parties' coins are treated as part of their input. Since our theorem quantifies over all inputs x, this will also capture the case where part of the input (corresponding to the random tapes of the randomized protocol) is uniformly sampled.

Let (C, P) be the circuit representation of Π. (Recall that C is Boolean circuit of size m that is obtained by gluing together the circuits of the next-round functions of each party via simple transmission gates, and that $P : [m] \to [n]$ is a mapping from the wires in C to the n parties; see Section 5.1.5.) The protocol $\hat{\Pi}^h$ is a non-interactive oracle-aided protocol, i.e., it contains a pre-processing step where each party locally computes a message to be sent to the oracle, followed by an oracle response and local post-processing.

- **Preprocessing.** Every party i samples random binary masks $\boldsymbol{\alpha}_i = (\alpha_j)_{j:P(j)=i}$ for each wire j that belongs to i, and for each of her input/local gates g computes the value $\mathbf{\Gamma}_g$ based on her input x_i and randomness $\boldsymbol{\alpha}_i$ as in Section 5.3. The party sends $\ell_i = (\boldsymbol{\alpha}_i, (\mathbf{\Gamma}_g))$ to the oracle h.

- **Oracle.** The oracle h takes all messages $\ell_i = (\boldsymbol{\alpha}_i, \mathbf{\Gamma}_i)$, concatenate them into $\mathbf{\Gamma} = (\mathbf{\Gamma}_g)_{g \in C} \circ \boldsymbol{\alpha}$, where $\boldsymbol{\alpha} = (\alpha_i)_{i \in [m]}$, and applies enc on $\mathbf{\Gamma}$ (and on a fresh random vector of field elements of appropriate length). The oracle sends the output $\mathbf{Q} = \mathsf{enc}^C(\mathbf{\Gamma})$ to all parties as a response to their query.

- **Post-processing.** Upon receiving $\mathbf{Q}$, each party i applies $\mathsf{dec}(\mathbf{Q})$ to obtain the sequence $(v_j)_{j \in [m]}$, where dec is the decoding algorithm, as described in Section 5.3. Then, for any output wire j belonging to party i, it computes $v_j - \alpha_j$ to obtain the output value (recall

[10] One can always replace h by a deterministic degree-2 function by applying the standard reduction [Gol04, Prop. 7.3.4] in which each random field element r is replaced by the sum $\sum_i r_i$ where r_i is a random field element selected by the i-th party as part of the preprocessing step. The resulting element r is uniformly distributed as long as at least one party is honest, and the transformation preserves the degree of the oracle.

that for a wire j belonging to party i, the value α_j was locally generated by party i and is therefore available for post-processing). Its output contains the collection of values on these output wires.

Properties 1 and 2 in the theorem follow immediately from the properties of the garbled-circuit encoding (Proposition 5.18). It remains to prove property 3.

Let $(\hat{\mathcal{A}}, J)$ be an adversary for $\hat{\Pi}^h$, where $J \subseteq [n]$ is the set of malicious parties. Since $\hat{\Pi}^h$ is non-interactive, then $\hat{\mathcal{A}}$ only gets to choose the values $\boldsymbol{\ell}[J] = \{\ell_i\}_{i\in J}$ based on the inputs $\mathbf{x}[J]$, and then post-process the oracle response $\mathbf{Q}$. We can further simplify and consider without loss of generality only adversaries $\hat{\mathcal{A}}$ that are deterministic (since our simulation is perfect and therefore holds even conditioned on any random string) and do not perform any post-processing but instead just output $\mathbf{Q}$ (since any post-processing results in a deterministic function of $\mathbf{Q}$, thus simulating $\mathbf{Q}$ allows to simulate any such value).

The simulator: overview. Our task is to produce an adversary $(\mathcal{A}, J)$ for the original protocol Π with the same real-model distribution as our (deterministic, no-post-processing) $\hat{\mathcal{A}}$. We assume throughout that $J \neq [n]$ (i.e. there exists some honest parties), or otherwise the result is trivial. In short, we use $\mathcal{A}$ to generate an I-gate vector $\boldsymbol{\Gamma}_I$ where I is the set of gates in C that belong to the coalition J. Then, invoke Lemma 5.20 to compute an effective I-corrupted circuit $\hat{C}$ together with effective inputs and masks $(\hat{\mathbf{x}}[I_{\mathsf{inp}}], \hat{\boldsymbol{\alpha}}[I_{\mathsf{wire}}])$. These values naturally define a cheating strategy for Π that will be played by $\mathcal{A}$. As a result, $\mathcal{A}$ learns the value $g_{\boldsymbol{\Gamma}_I}(\mathbf{x}[\bar{I}_{\mathsf{inp}}], \boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}])$ where $\mathbf{x}[\bar{I}_{\mathsf{inp}}], \boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}]$ are the inputs of the honest players (and $g_{\boldsymbol{\Gamma}_I}$ is defined in Lemma 5.20). This output is mapped to an h-output $\mathbf{Q}'$ via the simulator that is promised by Lemma 5.20. Details follow.

The admissible set I. For a malicious party p and round r, recall that $C_{r,p}$ is the set of gates corresponding to the computation of party p in the r-th round. Let I be the set of all gates in $C_{r,p}$ and transmission gates corresponding to the outputs of $C_{r,p}$, for all $p \in J$ and $r \in [R+1]$ where R is the number of rounds of the protocol. That is, I is the set of all gates that belong to the malicious parties. First note that I is an admissible set (see Section 5.3.2 for the definition). Indeed, every wire connecting a gate $g \in \bar{I}$ to a gate $g' \in I$ goes from transmission gate of round $r-1$ to a local gate of round r, and every wire connecting a gate $g \in I$ to a gate $g' \in \bar{I}$ goes from a transmission gate of round $r-1$ to a local gate of round r. Furthermore, note that the set of wires I_{wire} that are associated with I (see Section 5.3.2 for the definition) is exactly the set of all wires j such that $P(j) \in J$, i.e., all wires that belong to parties controlled by the adversary.

The adversary $\mathcal{A}$: Pre-execution. The adversary $\mathcal{A}$ first simulates $\hat{\mathcal{A}}$ on $\mathbf{x}[J]$ to obtain the values $\boldsymbol{\ell}[J]$ which define an I-gate vector $\boldsymbol{\Gamma}_I$. Then it applies the mapping T promised by Lemma 5.20 to $\boldsymbol{\Gamma}_I$ and gets a triple $(\hat{\mathbf{x}}[I_{\mathsf{inp}}], \hat{\boldsymbol{\alpha}}[I_{\mathsf{wire}}], \hat{C})$ of a binary I-input vector $\hat{\mathbf{x}}[I_{\mathsf{inp}}]$, a binary I-mask vector $\hat{\boldsymbol{\alpha}}[I_{\mathsf{wire}}]$, and an I-corrupted circuit $\hat{C}$ of C.

The adversary $\mathcal{A}$: Executing $\hat{C}$. The next step is to simulate the computation of $\hat{C}(\hat{\mathbf{x}}[I_{\mathsf{inp}}], \mathbf{x}[\bar{I}_{\mathsf{inp}}])$ by simulating the circuit representation C of the protocol with some malicious strategy. (Recall that we denote by $\mathbf{x}[\bar{I}_{\mathsf{inp}}]$ the vector of all inputs in $\mathbf{x}$ that correspond to input gates in $\bar{I}$. In our context $\mathbf{x}[\bar{I}_{\mathsf{inp}}]$ is the same as $\mathbf{x}[\overline{J}]$.) Since $\hat{C}$ is an I-corrupted version of I, the emulation is straightforward. Whenever the protocol Π represented by C instructs the adversary

to use the i-th input in some input gate $g \in I$, the adversary uses the effective bit $\hat{\mathbf{x}}[i]$. Whenever the protocol Π instructs the adversary to make a local computation step that corresponds to a gate g, the adversary follows the corresponding (modified) gate g in $\hat{C}$. Finally, when the protocol instructs the adversary to transmit a message u_c (either a private message or a broadcast message) according to a transmission gate $g \in I$ with an incoming wire c, the adversary acts according to the corresponding generalized transmission gate in $\hat{C}$. That is, for an out-going wire w the adversary sends the value $u_w = \hat{a} \cdot u + \hat{b}$ where $\hat{a}, \hat{b}$ are the values that appear next to the wire w in the generalized transmission gate of $\hat{C}$. Since the generalized transmission gates are $\bar{I}$-consistent all honest parties get the same value and so this step is realizable even when the protocol Π makes use of a broadcast channel.

The adversary $\mathcal{A}$: Post-execution. After the execution, the adversary collects the binary values of all wires that belong to the adversary, denoted by $\mathbf{u}[I_{\mathsf{wire}}]$, masks them with the vector $\hat{\boldsymbol{\alpha}}[I_{\mathsf{wire}}]$ that was generated before the execution, and generates the masked vector $\mathbf{v}[I_{\mathsf{wire}}] = \mathbf{u}[I_{\mathsf{wire}}] + \hat{\boldsymbol{\alpha}}[I_{\mathsf{wire}}]$. Then, the adversary uniformly samples a binary vector $\mathbf{v}[\bar{I}_{\mathsf{wire}}]$ of binary (masked) values for wires in $\bar{I}_{\mathsf{wire}}$, feeds the vector $(\mathbf{v}[I_{\mathsf{wire}}], \mathbf{v}[\bar{I}_{\mathsf{wire}}])$ to the simulator $S_{\boldsymbol{\Gamma}_I}$ of the randomized encoding promised in Lemma 5.20, and terminates with the resulting output.

Analyzing $\mathcal{A}$. The efficiency of $\mathcal{A}$ follows from Lemma 5.20. Moreover, the second item of the lemma implies that a semi-honest adversary $\hat{\mathcal{A}}$ that uses inputs $\mathbf{x}[I_{\mathsf{inp}}]$ induces a semi-honest adversary $\mathcal{A}$ since $\hat{C} = C$ and $\hat{\mathbf{x}}[I_{\mathsf{inp}}] = \mathbf{x}[I_{\mathsf{inp}}]$. It remains to show that $\mathsf{Real}_{\Pi,J,\mathcal{A}}(\mathbf{x}) \equiv \mathsf{Real}_{\hat{\Pi}^h,J,\hat{\mathcal{A}}}(\mathbf{x})$ for every $\mathbf{x}$. Recall that each of these random variables consist of two parts: (1) The adversary's output which is a vector of garbled gate tables of the same syntax as the output of enc; and (2) The output vector of the honest parties in which each entry corresponds to an output wire in C that belongs to an honest party. Recall that this set of output wires is denoted by $\bar{I}_{\mathsf{out}} \subset \bar{I}_{\mathsf{wire}}$.

Let us fix the value $\mathbf{x}$ throughout the proof. Since $\hat{\mathcal{A}}$ is deterministic, this also fixes the vector $\boldsymbol{\Gamma}_I$, the effective input $\hat{\mathbf{x}}[I_{\mathsf{inp}}]$, the masks $\hat{\boldsymbol{\alpha}}[I_{\mathsf{wire}}]$ and the effective circuit $\hat{C}$. Let us take a closer look at $\mathsf{Real}_{\Pi,J,\mathcal{A}}(\mathbf{x})$. Denote by $\mathbf{u} = (u_i)_{i\in[m]}$ the vector of all intermediate wire values induced by the evaluation of $\hat{C}(\hat{\mathbf{x}}[I_{\mathsf{inp}}], \mathbf{x}[\bar{I}_{\mathsf{inp}}])$. Since the adversary $\mathcal{A}$ executes the protocol represented by $\hat{C}(\hat{\mathbf{x}}[I_{\mathsf{inp}}], \mathbf{x}[\bar{I}_{\mathsf{inp}}])$, the output of the honest parties is $\mathbf{u}[\bar{I}_{\mathsf{out}}]$. Let $\boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}] = \mathbf{v}[\bar{I}_{\mathsf{wire}}] - \mathbf{u}[\bar{I}_{\mathsf{wire}}]$ where $\mathbf{v}[\bar{I}_{\mathsf{wire}}]$ is the vector chosen by the simulator in the post-execution step. We can now write $\mathsf{Real}_{\Pi,J,\mathcal{A}}(\mathbf{x})$ as

$$\left(\mathbf{v}[\bar{I}_{\mathsf{out}}] - \boldsymbol{\alpha}[\bar{I}_{\mathsf{out}}], S_{\boldsymbol{\Gamma}_I}(\mathbf{v})\right),$$

where

$$\mathbf{v} = \mathsf{gEval}_{\hat{C}}\left((\hat{\mathbf{x}}[I_{\mathsf{inp}}], \mathbf{x}[\bar{I}_{\mathsf{inp}}]), (\hat{\boldsymbol{\alpha}}[I_{\mathsf{wire}}], \boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}])\right),$$

and $\mathsf{gEval}_{\hat{C}}$ is the garbled-evaluation function.

On the other hand, the distribution $\mathsf{Real}_{\hat{\Pi}^h,J,\hat{\mathcal{A}}}(\mathbf{x})$ can be written as

$$\left(\mathsf{dec}(\mathbf{Q})[\bar{I}_{\mathsf{out}}] - \boldsymbol{\alpha}[\bar{I}_{\mathsf{out}}], \mathbf{Q}\right),$$

where

$$\mathbf{Q} = \mathsf{enc}^C\left(\boldsymbol{\Gamma}_I, \mathsf{pre}^C_{\bar{I}}(\mathbf{x}[\bar{I}_{\mathsf{inp}}], \boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}])\right),$$

and $\boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}]$ is uniformly chosen. Observe that the marginal distribution of $\boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}]$ in both experiments is uniform. Let us condition on the event that these two random variables take the same

fixed value. Then, $\mathsf{Real}_{\Pi,J,\mathcal{A}}(\mathbf{x})$ is distributed identically to $\mathsf{Real}_{\hat{\Pi}^h,J,\hat{\mathcal{A}}}(\mathbf{x})$ since, by Lemma 5.20, $\mathbf{Q}$ perfectly encodes $\mathbf{v}$. The theorem follows. □

5.5 Completeness Theorems

In this section we show that degree-2 functionalities are complete under non-interactive reductions. We say that a protocol has a security loss of L if any viable real-world adversary $\mathcal{A}$ can be simulated by an ideal-world adversary $\mathcal{B}$ whose complexity is at most L times larger than the complexity of $\mathcal{A}$. We prove the following completeness theorem (whose informal version appears as Theorem 1.5).

Theorem 5.23 (Completeness of quadratic functions)**.** *Let f be an n-party functionality computable by a Boolean circuit of size S and depth D and let $k > 1$ be an integer. Then there exists a non-interactive reduction from the task of securely computing f to the task of computing a degree-2 functionality over the field $\mathbb{F}_{2^k}$. The reduction can take any of the following forms:*

1. *Perfectly-secure reduction with threshold of $t = \lceil \frac{n}{3} - 1 \rceil$ whose computational complexity and security loss are $\mathrm{poly}(n, S, 2^D, k)$.*
2. *Statistically-secure reduction with threshold of $t = \lceil \frac{n}{2} - 1 \rceil$ whose computational complexity and security loss are $\mathrm{poly}(n, S, 2^D, k)$.*
3. *Assuming one-way functions, computationally secure reduction with threshold of $t = \lceil \frac{n}{2} - 1 \rceil$ whose computational complexity and security loss are $\mathrm{poly}(n, S, k)$. Furthermore, the reduction makes a black-box use of the one-way function (as part of the preprocessing and post-processing phases).*[11]

An identical statement for the special case of the binary field ($k = 1$) appears in [ABT19]. We mention again that asymptotically when $f = \{f_\kappa\}$ is an infinite functionality the parameters n, S, D and k are all functions of κ.

Proof. The proof of the theorem is obtained by plugging into our new master-theorem (Theorem 5.21) standard protocols from the literature. Specifically, the completeness proof of [ABT19, Section 5] shows that there exist perfectly-secure protocol with $t = \lceil \frac{n}{3} - 1 \rceil$ whose circuit complexity is $\mathrm{poly}(n, S, 2^D)$, a statistically-secure protocol with $t = \lceil \frac{n}{2} - 1 \rceil$) whose circuit complexity is $\mathrm{poly}(n, S, 2^D)$ and, assuming one-way functions, a computationally-secure protocol that makes a black-box use of the one-way function, whose circuit complexity is $\mathrm{poly}(n, S, k)$. □

We make few comments about the protocols obtained in Theorem 5.23 which are taken verbatim from [ABT19]. The protocols are employed over synchronous network with pairwise private channels and a broadcast channel (which is our default setting). In all three settings, we require full security (in particular, the adversary cannot abort the honest parties). It is well known that in this case the best achievable threshold is $\lceil (n/3) - 1 \rceil$ for perfect MPC [BGW88] and $\lceil (n/2) - 1 \rceil$ for statistical, or even computational MPC [RB89]. Hence, the theorem achieves optimal security thresholds in all three cases.

As usual in the context of constant-round information-theoretic MPC, our information-theoretic protocols are efficient only for NC^1 functionalities.[12] Nevertheless, even for general functions, for

[11] In the computational setting, we let the circuit size S play the role of the security parameter, and assume that n is at most polynomial in S.

[12] This can be slightly pushed to log-space computation via standard techniques.

which our perfect and statistical reductions are inefficient, the result remains meaningful since the protocols resist computationally unbounded adversaries.

References

[ABT18] Benny Applebaum, Zvika Brakerski, and Rotem Tsabary. Perfect secure computation in two rounds. In *Theory of Cryptography - 16th International Conference, TCC 2018, Panaji, India, November 11-14, 2018, Proceedings, Part I*, pages 152–174, 2018.

[ABT19] Benny Applebaum, Zvika Brakerski, and Rotem Tsabary. Degree 2 is complete for the round-complexity of malicious MPC. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*, pages 504–531, 2019.

[ACGJ18] Prabhanjan Ananth, Arka Rai Choudhuri, Aarushi Goel, and Abhishek Jain. Round-optimal secure multiparty computation with honest majority. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, pages 395–424, 2018.

[ACGJ19] Prabhanjan Ananth, Arka Rai Choudhuri, Aarushi Goel, and Abhishek Jain. Two round information-theoretic MPC with malicious security. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*, pages 532–561, 2019.

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC0. In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 166–175, 2004.

[AIK14] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. *SIAM J. Comput.*, 43(2):905–929, 2014.

[AL17] Gilad Asharov and Yehuda Lindell. A full proof of the BGW protocol for perfectly secure multiparty computation. *J. Cryptology*, 30(1):58–151, 2017.

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

[BB89] Judit Bar-Ilan and Donald Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing, Edmonton, Alberta, Canada, August 14-16, 1989*, pages 201–209, 1989.

[Bea91] D. Beaver. Efficient Multiparty Protocols Using Circuit Randomization. In J. Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer Verlag, 1991.

[BFKR90] Donald Beaver, Joan Feigenbaum, Joe Kilian, and Phillip Rogaway. Security with low communication overhead. In *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, pages 62–76, 1990.

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

[BH08] Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure MPC with linear communication complexity. In *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008*, pages 213–230, 2008.

[BKP11] Michael Backes, Aniket Kate, and Arpita Patra. Computational verifiable secret sharing revisited. In *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, pages 590–609, 2011.

[BL18] Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, pages 500–532, 2018.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 503–513, 1990.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145, 2001.

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

[CD01] Ronald Cramer and Ivan Damgård. Secure distributed linear algebra in a constant number of rounds. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, pages 119–136, 2001.

[CGMA85] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pages 383–395, 1985.

[CKPR01] Ran Canetti, Joe Kilian, Erez Petrank, and Alon Rosen. Black-box concurrent zero-knowledge requires omega~(log n) rounds. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 570–579, 2001.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 364–369, 1986.

[DI05] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, pages 378–394, 2005.

[DR85] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *J. ACM*, 32(1):191–204, 1985.

[FGG+06] Matthias Fitzi, Juan A. Garay, Shyamnath Gollakota, C. Pandu Rangan, and K. Srinathan. Round-optimal and efficient verifiable secret sharing. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, pages 329–342, 2006.

[FM85] Paul Feldman and Silvio Micali. Byzantine agreement in constant expected time (and trusting no one). In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pages 267–276, 1985.

[GIKR01] Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. The round complexity of verifiable secret sharing and secure multicast. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 580–589, 2001.

[GIKR02] Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. On 2-round secure multiparty computation. In *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, pages 178–193, 2002.

[GIS18] Sanjam Garg, Yuval Ishai, and Akshayaram Srinivasan. Two-round MPC: information-theoretic and black-box. In *Theory of Cryptography - 16th International Conference, TCC 2018, Panaji, India, November 11-14, 2018, Proceedings, Part I*, pages 123–151, 2018.

[GK96] Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM J. Comput.*, 25(1):169–192, 1996.

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications.* Cambridge University Press, 2004.

[GS18] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, pages 468–499, 2018.

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 294–304, 2000.

[IK02] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, pages 244–256, 2002.

[IKKP15] Yuval Ishai, Ranjit Kumaresan, Eyal Kushilevitz, and Anat Paskin-Cherniavsky. Secure computation with minimal interaction, revisited. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 359–378, 2015.

[IKP10] Yuval Ishai, Eyal Kushilevitz, and Anat Paskin. Secure multiparty computation with minimal interaction. In *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, pages 577–594, 2010.

[KKK09] Jonathan Katz, Chiu-Yuen Koo, and Ranjit Kumaresan. Improving the round complexity of VSS in point-to-point networks. *Inf. Comput.*, 207(8):889–899, 2009.

[KLR06] Eyal Kushilevitz, Yehuda Lindell, and Tal Rabin. Information-theoretically secure protocols and security under composition. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 109–118, 2006.

[KPR10] Ranjit Kumaresan, Arpita Patra, and C. Pandu Rangan. The round complexity of verifiable secret sharing: The statistical case. In *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, pages 431–447, 2010.

[LF82] Leslie Lamport and Michael Fischer. Byzantine generals and transaction commit protocols. Technical report, Technical Report 62, SRI International, 1982.

[Lyn96] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[MNS16] Tal Moran, Moni Naor, and Gil Segev. An optimally fair coin toss. *J. Cryptology*, 29(3):491–513, 2016.

[PCRR09] Arpita Patra, Ashish Choudhary, Tal Rabin, and C. Pandu Rangan. The round complexity of verifiable secret sharing revisited. In *Advances in Cryptology - CRYPTO*

2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings, pages 487–504, 2009.

[PR18] Arpita Patra and Divya Ravi. On the power of hybrid networks in multi-party computation. *IEEE Trans. Information Theory*, 64(6):4207–4227, 2018.

[PSL80] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 73–85, 1989.

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

[SYY99] Tomas Sander, Adam L. Young, and Moti Yung. Non-interactive cryptocomputing for NC1. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 554–567, 1999.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 162–167, 1986.

A Lower Bound

A.1 Perfectly-Secure VSS

Definition A.1 (**Perfectly-secure VSS [CGMA85]**). *In a VSS protocol there is a distinguished party $D \in \mathcal{P}$ that holds an input s picked from any distribution over a field $\mathbb{F}$ referred to as the secret. The protocol consists of two phases, a* sharing *phase and a* reconstruction *phase.*

- *Sharing: In the beginning, D holds s and each party including the dealer holds an independent random input r_i. The sharing phase may span over several rounds. At each round, each party can privately send messages to the other parties and it can also broadcast a message. Each message sent or broadcasted by P_i is determined by its input (if any), its random input and messages received from other parties in previous rounds.*
- *Reconstruction: In this phase, each P_i provides its entire view v_i from the sharing phase, and a reconstruction function is applied and is taken as the protocol's output.*

A two-phase, n party protocol as above is called a perfectly-secure (n, t)-VSS, if for any adversary $\mathcal{A}$ corrupting at most t parties, the following holds:

- *Correctness: If D is honest then each honest party upon completing the reconstruction phase, outputs s.*
- *Commitment: Even if D is corrupt, any execution of the sharing phase determines a unique value $\bar{s}$, such that each honest party upon completing reconstruction phase outputs $\bar{s}$, irrespective of the behavior of the corrupted parties.*

- *Privacy: If D is honest then the adversary's view during the sharing phase reveals no information on s. More formally, the adversary's view is identically distributed for all possible values of s. Denoting $\mathcal{D}_s$ as $\mathcal{A}$'s view during the sharing phase when D's secret is s, the privacy property demands $\mathcal{D}_s \equiv \mathcal{D}_{s'}$ for any $s \neq s'$.*

A.2 General transference claim

We prove that nay RT T that is i-realizable in π', is also i-realizable in π.

Claim A.2 (transference claim)**.** *If T is i-realizable in π' then it is also i-realizable in π.*

Proof. Given a RT $T = (\mathbf{v}, \mathbf{a}, \mathbf{b})$ and index $i \in [4]$, we define the (T, i)-canonical adversary P_i^* in π as follows. Let $\mathbf{c}$ be the vector of broadcasts of the third round which is induced by T.

1. (Round 1) Broadcast a_i and private messages $a_{i,j}$ as defined by T;
2. (Round 2) Broadcast b_i;
3. (Round 3) Broadcast c_i.

We show that the (T, i)-canonical adversary realizes T in π. Consider an execution of π in which the honest parties have inputs $(z_j)_{j \neq i}$ and pick randomness $(r_j)_{j \neq i}$, where z_j and r_j are the input and randomness of party j, as induced by T. By definition, the inputs/randomness of every honest party $j \neq i$ is consistent with v_j as defined by T. Since the view of each honest party j is self consistent (by Observation 3.10) its first-round broadcast and all the first round outgoing messages of j to an honest party $\ell \neq i$, are consistent with the transcript values $(a_j, (a_{j,\ell})_{\ell \neq i,j})$. Consequently, every message that an honest party j receives from an honest party ℓ in the first round is consistent with T (since honest parties are pairwise consistent). In addition, the incoming message from P_i^* is consistent with T by definition of the canonical adversary.

Similarly, the broadcast messages that an honest party j sends in the second round are consistent with T since v_j is self-consistent in T, the broadcast value is computed based on P_j's inputs, randomness, and incoming messages which are all consistent with T. Therefore, every broadcast message that an honest party j receives from an honest party ℓ in the first round is consistent with T (since honest parties are pairwise consistent). In addition, the broadcast message from P_i^* is consistent with T by the definition of the canonical adversary.

Finally, the broadcast messages that an honest party j sends in the second round are consistent with T since $\alpha_j = 0$ and so the value c_j induced by T is computed just like in π based on values that were computed in previous rounds, and party i broadcasts the value c_i which is induced by T. □

B Perfectly-secure MPC

Security of all the protocols are proven based on real and ideal world paradigm, as recalled in Section 5.1.4.

B.1 Weak Commitment

Proof. We divide the proof into two cases based on whether D is corrupt or not. We design simulator and show indistinguishability for each case.

Honest D. The simulator invokes $\mathcal{A}$ with its auxiliary information z. It then receives s_i for every corrupt P_i in set I from functionality $\mathcal{F}_{\mathsf{wc}}$ and picks a $g'(x)$ polynomial such that $g'(i) = s_i$ for every $P_i \in I$. It then picks a symmetric bivariate polynomial $G'(x, y)$ uniformly at random with $G'(x, 0) = g'(x)$ and emulates the role of the honest parties to the set of corrupt parties in I. The simulator outputs what $\mathcal{A}$ outputs.

Denoting $g'_i(x) = G'(x, i)$, the set polynomials $\{g'_i(x)\}_{i \in I}$ determine the values that the simulator on behalf of the honest D is supposed to disclose for the pairs whose masked values do not match. This is because no two honest parties conflict over their masked values. The same is true even for a real execution. The only difference between the real and simulated execution is whether $\{g'_i(x)\}_{i \in I}$ is based on the real polynomial $g(x)$ or the simulator-chosen polynomial $g'(x)$. These distributions $\{g_i(x)\}_{i \in I}$ and $\{g'_i(x)\}_{i \in I}$ are indeed identical. A formal argument for this can be derived from the argument given in Claim 5.4 of [AL17] for non-symmetric bivariate polynomials. It is easy to see that the outputs of the honest parties in $\mathcal{P} \setminus I$ are the same in both the worlds for every input $g(x)$ and auxiliary input of the adversary $\mathcal{A}$. Therefore the simulation ensures that the joint distribution of the views of honest (which are their outputs) and corrupt parties have identical distribution in both the worlds over the random coins of the honest parties, for every input $g(x)$ and auxiliary input of the adversary $\mathcal{A}$ (which can be assumed to hold the best coins of adversary). This completes the proof for honest D.

Corrupt D. The simulator invokes $\mathcal{A}$ with its auxiliary information z. The simulator plays the role of the $n - |I|$ honest parties and receives the $g'_i(x)$ polynomials from corrupt D. It then emulates the role of the honest parties using these polynomials. At the end of the simulated protocol execution, if D is discarded, then it sends x^{2t} and W to $\mathcal{F}_{\mathsf{wc}}$. Sending x^{2t} ensures that the functionality sends $\perp$ to all the honest parties. Otherwise, let H be the set of honest parties contained in W. Since the $g'_i(x)$ polynomials possessed by the parties in H are consistent with each other, they define a unique symmetric bivariate polynomial of degree at most t in both x and y, say $G'(x, y)$. Let $g'(x) = G'(x, 0)$. The simulator forwards $g'(x)$ and W to $\mathcal{F}_{\mathsf{wc}}$ and outputs what $\mathcal{A}$ outputs. This completes the simulation.

Since the simulator emulates the honest parties as per the specification of the protocol, the simulated view of the corrupt parties is perfectly indistinguishable from their real world view. Assuming (a) the random coins of the corrupt parties come from $\mathcal{A}$'s auxiliary input and (b) the simulator and the honest parties in the real world pick the same randomness, the adversary sees the *identical* view across the real and simulated world for every choice of its input $g'(x)$. We now show that the outputs of the honest parties in $\mathcal{P} \setminus I$ are the same in both the worlds for every input $g'(x)$, auxiliary input of the adversary $\mathcal{A}$, and the random coins chosen for the honest parties. The entire execution in both the worlds, under the above circumstances, turn identical which readily implies that the set W is identical in both worlds. From a real execution, it is clear that all the honest parties in W are pairwise consistent implying their polynomials uniquely determine a bivariate polynomial $G'(x, y)$ and its underlying univariate polynomial $G'(x, 0) = g'(x)$. As a result, in the real execution, every honest party in W outputs $g'(i)$ and others output $\perp$. In the simulated world, the simulator, on seeing the same W, computes $G'(x, y)$ and $g'(x)$ with the honest parties polynomials in W. Since it forwards $(g'(x), \mathsf{W})$ to $\mathcal{F}_{\mathsf{wc}}$, honest parties in W receive $g'(i)$ from the functionality, while the honest parties outside receive $\perp$. Therefore, the outputs of the honest parties are identical in both the worlds for every input $g'(x)$, auxiliary input of the adversary $\mathcal{A}$, and the random coins chosen for the honest parties. This implies that the joint distribution of the views of honest (which are

their outputs) and corrupt parties have identical distribution in both the worlds over the random coins of the honest parties, for every input $g(x)$ and auxiliary input of the adversary $\mathcal{A}$. □

B.2 Verifiable Secret Sharing (VSS)

Proof. We divide the proof into two cases based on whether D is corrupt or not. We design simulator and show indistinguishability for each case.

Honest D. The simulator $\mathcal{V}^h$ initiates $\mathcal{A}$ with auxiliary input z and performs three types of communication– (i) external communication to functionality $\mathcal{F}_{\mathsf{vsh}}$, (ii) simulation of wcom_i via invoking a copy of a simulator for wcom, denoted as $\mathcal{W}_i$ (which is either for the honest or corrupt dealer case based on whether P_i is honest or corrupt) and (iii) simulation of the honest parties to $\mathcal{A}$ as per the steps of protocol vsh outside the executions $\{\mathsf{wcom}_i\}_{i\in\{1,\ldots,n\}}$. On receiving polynomials $\{f_i(x)\}_{i\in I}$ from $\mathcal{F}_{\mathsf{vsh}}$, $\mathcal{V}_h$ emulates the honest dealer by picking a symmetric bivariate polynomial $F(x,y)$ with the constraint $F(x,i) = f_i(x)$ for every $i \in I$. It internally invokes $\mathcal{W}_i$, a simulator for honest dealer case, for $i \notin I$ with t values $\{h_{ij}\}_{j\in I}$ chosen uniformly at random. Let $h_i(x)$ is the polynomial that $\mathcal{W}_i$ uses to simulate wcom_i. $\mathcal{V}^h$ uses $h_i(x)$ as the blinder polynomial of honest P_i to compute the blinded polynomial on behalf of the honest parties. For every wcom_i, $i \in I$, it runs $\mathcal{W}_i$, a simulator for corrupt dealer case. Let $\{h_{ij}\}_{j\notin I}$ are values that $\mathcal{W}_i$ receives from $\mathcal{A}$ in wcom_i. $\mathcal{V}_h$ uses these as the blinders the honest parties received from corrupted P_i in wcom_i and simulates the role for the honest parties in vsh. The simulator outputs what $\mathcal{A}$ outputs. This concludes the simulation. The outputs of the honest parties in the two worlds are identical for every input $F(x,y)$ and auxiliary input of the adversary $\mathcal{A}$. Below we argue the indistinguishability of views of $\mathcal{A}$ in the two worlds.

First, we note that the simulator makes sure, just like the real world, that the adversary first sees its messages in Round 1 of vsh and its messages in all wcom_i for honest P_i, before it distributes the shares of the blinder polynomials inside wcom of the corrupt parties. Given this, the simulations of all wcom_i is perfectly indistinguishable from the real world. Further, the polynomials corresponding to the corrupt parties, $\{f_i(x)\}_{i\in I}$, entirely determine the values that the simulator on behalf of the honest D is supposed to disclose for pairs whose pairwise consistency check fails (those who are in conflict). The same is true even for a real execution. The only difference between the real and simulated execution is whether $\{f_i(x)\}_{i\in I}$ is based on the real polynomial $F'(x,y)$ or the simulator-chosen polynomial $F(x,y)$. These distributions $\{f'_i(x)\}_{i\in I}$ and $\{f_i(x)\}_{i\in I}$ are indeed identical. A formal argument for this can be derived from the argument given in Claim 5.4 of [AL17] for non-symmetric bivariate polynomials. This completes the proof for honest D.

D is corrupt. The simulator $\mathcal{V}^c$ initiates $\mathcal{A}$ with auxiliary input z and performs three types of communication as mentioned in the previous case. The invocations to $\{\mathcal{W}_i\}$ are handled in the same way and same order i.e. $\{\mathcal{W}_i\}_{i\notin I}$ are invoked first so that $\mathcal{A}$ sees its messages first in these instances, followed by the invocations to $\{\mathcal{W}_i\}_{i\in I}$. Next, $\mathcal{V}^c$ perfectly simulates the steps of vsh on behalf of the honest parties on receiving $\{f_i(x)\}_{i\notin I}$ from $\mathcal{A}$. In the end, if D is discarded, then send $x^{2t}y^{2t}$ to functionality $\mathcal{F}_{\mathsf{vsh}}$ so that it sets $F(x,y)$ to a default symmetric bivariate polynomial of degree t in both variables and distributes the shares accordingly. Otherwise, $\mathcal{V}^c$ computes $F(x,y)$ as the unique polynomial inferred by the $f_i(x)$ polynomials of the honest parties in V. Note that there exist a unique such polynomial since no honest parties in V conflict over their common shares ($f_i(j)$ and $f_j(i)$ respectively). $\mathcal{V}^c$ sends $F(x,y)$ to $\mathcal{F}_{\mathsf{vsh}}$. The simulator outputs what $\mathcal{A}$ outputs.

Since the simulator emulates the honest parties as per the protocol specification, the simulated view of the adversary $\mathcal{A}$ is indistinguishable from its real world view. Similar to the proof of corrupt dealer case for wcom, we can conclude that for a given $F(x,y)$, z (which includes the best coins of adversary and given this, $\mathcal{A}$ turns a deterministic algorithm), and the random coins for the honest parties, the the outputs of the honest parties in both world will be identical. In the real world, every honest P_i receives $f_i(x) = F(x,i)$. The same is true in the simulated world, as the simulator extracts the unique $F(x,y)$ from the polynomials of the the honest parties in V (which uniquely define $F(x,y)$) and forwards the same to $\mathcal{F}_{\mathsf{vsh}}$. This ensures all the honest parties receive $f_i(x) = F(x,y)$.This further implies the joint distribution of the outputs of honest parties and views of the corrupt parties in both the worlds are identical, for a given input $F(x,y)$ and auxiliary input z.

□

B.3 Multiplication Triple Sharing

Proof of Lemma 4.12. We divide the proof in two cases as before based on whether D is honest or corrupt. We design simulator and show indistinguishability for each case.

D is honest. The simulator $\mathcal{M}^h$ runs the adversary $\mathcal{A}$ with auxiliary input z. On receiving three set of values $\{a_i, b_i, c_i\}_{i\in I}$ from $\mathcal{F}_{\mathsf{msh}}$, it picks a set of $t+3$ random polynomials $\bar{f}^a(x), \bar{f}^b(x), \bar{f}^c(x), \bar{f}^1(x), \ldots, \bar{f}^t(x)$ as follows: (i) the polynomials $\bar{f}^a(x), \bar{f}^b(x), \bar{f}^c(x)$ are such that $\bar{f}^a(i) = a_i, \bar{f}^b(i) = b_i, \bar{f}^c(i) = c_i$ and their constant terms are equal to three random values $\bar{a}, \bar{b}, \bar{c}$ satisfying $\bar{c} = \bar{a}\bar{b}$; (ii) $\bar{f}^c(x) = \bar{f}^a(x)\bar{f}^b(x) - \sum_{\alpha=1}^{t} x^\alpha \bar{f}^\alpha(x)$. Next, following an honest dealer $\mathcal{M}^h$ picks symmetric bivariate polynomials $\bar{F}^a(x,y), \bar{F}^b(x), \bar{F}^c(x), \bar{F}^1(x), \ldots, \bar{F}^t(x)$ hiding the respective univariate polynomials in them. It then internally initiates $t+3$ VSS simulators for honest dealer case, denoted as $\mathcal{V}^a, \mathcal{V}^b, \mathcal{V}^c, \{\mathcal{V}^\alpha\}_{\alpha\in\{1,\ldots,t\}}$ with inputs respectively $\{\bar{f}_i^a(x)\}_{i\in I}$, $\{\bar{f}_i^b(x)\}_{i\in I}$, $\{\bar{f}_i^c(x)\}_{i\in I}$ for first three and with input $\{\bar{f}_i^\alpha(x)\}_{i\in I}$ for $\mathcal{V}^\alpha$ and making sure the bivariate polynomials used by these simulators are as chosen by her. Next, $\mathcal{V}^a, \mathcal{V}^b, \mathcal{V}^c, \{\mathcal{V}^\alpha\}_{\alpha\in\{1,\ldots,t\}}$ emulates their instances of VSS honestly. $\mathcal{M}^h$ emulates the honest parties role in the protocol outside the VSS instances. In the end, $\mathcal{M}^h$ outputs what the adversary outputs.

It is clear from the way the polynomials $\bar{f}^a(x), \bar{f}^b(x), \bar{f}^c(x), \bar{f}^1(x), \ldots, \bar{f}^t(x)$ are picked that $\bar{f}^a(x), \bar{f}^b(x), \bar{f}^c(x)$ are chosen uniformly at random with the constraint $f^c(0) = f^a(0)f^b(0)$. Now following security of VSS (which implies indistinguishability of the views inside the VSS instances), $\mathcal{A}$'s views across the two worlds are perfectly indistinguishable.

D is corrupt. The simulator for the corrupt dealer case, $\mathcal{M}^c$ runs the adversary $\mathcal{A}$ with auxiliary input z. On receiving, the $t+3$ set of polynomials $\{\bar{f}_i^a(x)\}_{i\notin I}$, $\{\bar{f}_i^b(x)\}_{i\notin I}$, $\{\bar{f}_i^c(x)\}_{i\notin I}$, $\{\bar{f}_i^\alpha(x)\}_{i\notin I}$ for $\alpha \in \{1,\ldots,t\}$, internally initiates $t+3$ VSS simulators for corrupt dealer case, $\mathcal{V}^a, \mathcal{V}^b, \mathcal{V}^c, \{\mathcal{V}^\alpha\}_{\alpha\in\{1,\ldots,t\}}$ with their respective set of polynomials. $\mathcal{M}^c$ then runs the execution and takes care of the steps outside the VSS instances as per protocol specification. If D is not discarded, then it extracts the unique bivariate polynomials $\bar{F}^a(x,y), \bar{F}^b(x), \bar{F}^c(x), \bar{F}^1(x), \ldots, \bar{F}^t(x)$ defined by the honest parties in V (they are guaranteed to exist as the security proof for VSS suggests). It then returns $\bar{f}^a(x) = \bar{F}^a(x,0)$, $\bar{f}^b(x) = \bar{F}^b(x,0)$ and $\bar{f}^c(x) = \bar{F}^c(x,0)$ to $\mathcal{F}_{\mathsf{msh}}$. Otherwise, it sends (x^{2t}, x^{2t}, x^{2t}) to $\mathcal{F}_{\mathsf{msh}}$ that, on seeing polynomials of degree more than t, distributes $\perp$ to the honest parties.

The distribution of the views of $\mathcal{A}$ in both the worlds are identical as $\mathcal{M}^c$ emulates the honest parties as per protocol specification. As per the arguments made in the proof of VSS for the corrupt dealer case, the outputs of the honest parties are identical in both the worlds when we fix the inputs $f_a(x), f_b(x), f_c(x)$, the auxiliary input z of $\mathcal{A}$ and pick the same randomness for the honest parties in the VSS instances. We now prove that $f^c(0) = f^a(0)f^b(0)$ when D is not discarded in the real world which will conclude the same for the simulated world. Let $f^a(x), f^b(x), f^c(x), f^1(x), \ldots, f^t(x)$ are the polynomials that are t-shared in the end of Round 3 (the guarantee that these are polynomials of degree t comes from the success of VSS when D is not discarded). Our protocol ensures that these polynomials satisfy $f^c(i) = f^a(i)f^b(i) - \sum_{\alpha=1}^{t} i^\alpha f^\alpha(i)$ for every honest P_i whether it belongs to V or not. In the former case, it is done via private check and for the latter it is done via public verification. That is, at least $2t+1$ parties confirm that this relation is satisfied. Now consider the two polynomials $f^c(x)$ and $f^a(x)f^b(x) - \sum_{\alpha=1}^{t} x^\alpha f^\alpha(x)$. The latter is a polynomial of degree at most $2t$ with constant term as ab. The former is a polynomial of degree at most t with constant term as c. Since these two polynomials intersect at $2t+1$ points, it is guaranteed that the polynomials are equal and the unique polynomial has degree at most t and has ab in the constant term.

When D is discarded, either one of the following is true: (a) polynomials $f^a(x), f^b(x), f^c(x), f^1(x), \ldots, f^t(x)$ are not of degree t; (b) the relation $f^c(i) = f^a(i)f^b(i) - \sum_{\alpha=1}^{t} i^\alpha f^\alpha(i)$ does not hold true for $i \notin I$; (iii) a corrupt P_i complains and the relation $f^c(i) = f^a(i)f^b(i) - \sum_{\alpha=1}^{t} i^\alpha f^\alpha(i)$ does not hold true in Round 4. In both the worlds, either of the above leads to every honest party outputting $\perp$. It is clear from the steps of the real protocol, while in the simulated world, the simulator makes sure the same by sending (x^{2t}, x^{2t}, x^{2t}) to $\mathcal{F}_{\mathsf{msh}}$. This completes the proof. □

B.4 Degree-2 Computation

Proof of Theorem 4.13. The simulator $\mathcal{S}$ initiates $\mathcal{A}$ with auxiliary input z and performs three types of communication– (i) simulation of $\mathsf{vsh}^\alpha, \mathsf{vsh}^\beta, \{\mathsf{vsh}^i, \mathsf{msh}^i\}_{i\in\{1,\ldots,n\}}$ via invoking a copy of their simulators either for the honest or corrupt dealer case based on whether their dealers are honest or not, (ii) simulation of the honest parties to $\mathcal{A}$ as per the steps of protocol $\mathsf{d2c}$ outside the executions $\mathsf{vsh}^\alpha, \mathsf{vsh}^\beta, \{\mathsf{vsh}^i, \mathsf{msh}^i\}_{i\in\{1,\ldots,n\}}$ and (iii) external communication to functionality $\mathcal{F}_{\mathsf{d2c}}$.

– *Simulation of the* vsh *and* msh *instances with honest dealers.* First, corresponding to the honest parties in $\mathcal{P} \setminus I$, the simulator emulates the honest parties with inputs as 0 and the triples as suggested in the protocol (i.e randomly picked triples (a^i, b^i, c^i) satisfying product relation). It then invokes the simulators for honest dealer case for all instances of vsh with the set of t polynomials (corresponding to I) on the chosen bivariate polynomials. These simulators take care of the simulation of these instances for their entire run until Round 3. Similarly, it invokes the simulators for honest dealer case of msh instances with honest dealers with the shares on the univariate polynomials corresponding to I. These take care of the simulation of these msh instances for their entire run until the end.

– *Simulation of the* vsh *and* msh *instances with corrupt dealers.* Next, it initiates simulators for corrupt dealer case corresponding to instances of vsh and msh with corrupt dealers. If the simulator for a vsh instance returns $x^{2t}y^{2t}$ (implying D of this instance is discarded), it picks a default value for the input corresponding to this instance. Otherwise, it receives the extracted inputs of the corrupt parties from the vsh instances. If the simulator for a msh instance returns (x^{2t}, x^{2t}, x^{2t})

(implying D of this instance is discarded), it removes the dealer from its set L which is initialised to $\mathcal{P}$.

– *Simulation of steps of* $\mathsf{d2c}$ *outside the* vsh *and* msh *instances.* It emulates the steps outside $\mathsf{vsh}^{\alpha}, \mathsf{vsh}^{\beta}, \{\mathsf{vsh}^{i}, \mathsf{msh}^{i}\}_{i\in\{1,\ldots,n\}}$ honestly and emulates $\mathcal{F}_{\langle 0\rangle}$ correctly.

– *Interaction with* $\mathcal{F}_{\mathsf{d2c}}$. It sends the inputs (either the default or the extracted ones) to $\mathcal{F}_{\mathsf{d2c}}$ on behalf of the corrupt parties and receives the output y.

– *Simulation of Round 4.* The simulator $\mathcal{S}$ simulates the reconstructions as a part of Beaver's trick honestly for every P_i and updates L. Now on holding y and knowing the shares of y possessed by the corrupt parties in I, it fits a random degree $2t$ polynomial over these points. This polynomial determines the share y_i for every honest P_i. Now for every party P_i in L, it uses y_i and the share-shares of y_i held by the corrupt parties in I to interpolate a degree t polynomial. This polynomial determines the share-shares of y_i for all the honest parties. The simulator thus adjusts these shares and share-shares using the shares and share-shares corresponding to $\langle 0\rangle$ generated by itself, while emulating $\mathcal{F}_{\langle 0\rangle}$. The simulator now discloses these adjusted share-shares of the ith share of y for every $P_i \in \mathsf{L}$ as a part of rec protocols. This ensures that the adversary outputs y, as returned by the functionality. This completes the simulation.

We now argue perfect indistinguishability between the simulated and the real world views. The execution of msh instances are done identically in both the worlds. The prime difference between these worlds is in the inputs of the honest parties which are real in the protocol, but 0s in the simulation. Relying on the security of the vsh protocol, we claim indistinguishability of the views generated inside the concerned vsh instances. The communication regarding the sharing corresponding to 0 inputs that are part of Beaver's trick are indistinguishable, owing to the unknown and random triples. Indeed, only the values of the form $(x_i^{\alpha} - a^i)$ and their shares are revealed. Finally, the way the shares and share-shares of y corresponding to the honest parties are computed takes care of the fact that it indeed corresponds to $\langle y\rangle$, barring the fact that the underlying $2t$ degree polynomial may not be a random polynomial. This is settled by blinding this sharing of y with a random $\langle 0\rangle$. Therefore, the revealed second-level sharings corresponding to the parties in L and the underlying $2t$-degree polynomial are random subject to the fact that y lies in the constant term of the $2t$ degree polynomial. This completes the proof. □

C Analyzing the Garbled Circuit (Proof of Lemma 5.20)

We begin by presenting the mapping T. The details are somewhat tedious, and the definition becomes clearer after seeing the correctness and privacy proofs that appear in the subsequent sections (Section C.1 and Section C.2).

Construction C.1 (The mapping T)**.** *The mapping T receives as an input an I-gate vectors $\mathbf{\Gamma}_I = (\mathbf{\Gamma}_g)_{g\in I} \circ \boldsymbol{\alpha}[I_{\mathsf{wire}}]$ and outputs a triple $(\hat{\mathbf{x}}[I_{\mathsf{inp}}], \hat{\boldsymbol{\alpha}}[I_{\mathsf{wire}}], \hat{C})$ which is defined as follows.*

- *For every $j \in I_{\mathsf{wire}}$ we define the effective mask*

$$\hat{\alpha}_j = \begin{cases} \alpha_j, & \text{if } \alpha_j \in \{0,1\}, \\ 0, & \text{otherwise,} \end{cases}$$

and take $\hat{\boldsymbol{\alpha}}[I_{\mathsf{wire}}]$ to be the concatenation of all effective masks of wires in I_{wire}.

- *For every input gate $g \in I$ with outgoing wire k, input-value $\mathbf{\Gamma}_g = x_g$ and wire mask α_k, we define the effective input*

$$\hat{x}_g = \begin{cases} (x_g + \alpha_k) - \hat{\alpha}_k, & \text{if } x_g + \alpha_k \in \{0,1\}, \\ 0 - \hat{\alpha}_k, & \text{otherwise}, \end{cases}$$

and take $\hat{\mathbf{x}}[I_{\mathsf{inp}}]$ to be the concatenation of all effective inputs of input gates in I.

- *The circuit $\hat{C}$ is obtained from C by modifying the semantics of local computation gates and generalized transmission gates in I as follows.*

  A local computation gate $g \in I$, with incoming wires c, d and outgoing wire k, is replaced by a local computation gate that computes the following effective *Boolean operator $\hat{G}$ defined via*

$$\hat{G}(\beta_0, \beta_1) := \begin{cases} (\gamma_g^{\beta_0 + \hat{\alpha}_c, \beta_1 + \hat{\alpha}_d} + \alpha_k) - \hat{\alpha}_k, & \text{if } (\gamma_g^{\beta_0 + \hat{\alpha}_c, \beta_1 + \hat{\alpha}_d} + \alpha_k) \in \{0,1\}, \\ 0 - \hat{\alpha}_k, & \text{otherwise}, \end{cases}$$

  for every $\beta_0, \beta_1 \in \{0,1\}$.

  A generalized transmission gate $g \in I$ with incoming wire c, outgoing wires $k_1, \ldots, k_p$, and every $j \in [p]$ such that wire k_j has label (a, b), is replaced with a generalized transmission gate in which the effective label *of wire k_j, denoted by $(\hat{a}, \hat{b})$, is defined via*

$$(\hat{a}, \hat{b}) := \begin{cases} (a, b) & \text{if } a = 0, \\ (a, b) & \text{if } a = 1,\ k_j \in \bar{I}_{\mathsf{wire}} \text{ and } \alpha_c \in \{0,1\}, \\ (0, 0) & \text{if } a = 1,\ k_j \in \bar{I}_{\mathsf{wire}} \text{ and } \alpha_c \notin \{0,1\}, \\ (1, \hat{\alpha}_c - \alpha_c + b + \alpha_{k_j} - \hat{\alpha}_{k_j}) & \text{if } a = 1,\ k_j \in I_{\mathsf{wire}} \text{ and } \alpha_{k_j} - \alpha_c \in \{0,1\}, \\ (0, 0) & \text{if } a = 1,\ k_j \in I_{\mathsf{wire}},\ \alpha_c \notin \{0,1\}, \text{ and } \alpha_{k_j} \in \{0,1\}, \\ (a, b) & \text{if } a = 1,\ k_j \in I_{\mathsf{wire}},\ \alpha_c \in \{0,1\}, \text{ and } \alpha_{k_j} \notin \{0,1\}, \\ (0, 0) & \text{if } a = 1,\ k_j \in I_{\mathsf{wire}},\ \alpha_c, \alpha_{k_j} \notin \{0,1\}, \text{ and } \alpha_{k_j} - \alpha_c \notin \{0,1\}. \end{cases}$$

One can easily verify that the mapping T satisfies the following syntactic properties.

Proposition C.2. *The mapping T can be computed by making $O(m)$ arithmetic operations. Moreover, for every circuit C, admissible set I and input $\mathbf{\Gamma}_I$, the circuit $\hat{C}$ has the same topology as C and every generalized transmission gate g in C which is $\bar{I}$-consistent (that is, the labels of all outgoing wires in $\bar{I}_{\mathsf{wire}}$ are the same) is transformed into an $\bar{I}$-consistent generalized transmission gate as well. That is, $\hat{C}$ is an I-corrupted version of C. Finally, for every* binary *I-input vector $\mathbf{x}[I_{\mathsf{inp}}]$ and* binary *I-mask vector $\boldsymbol{\alpha}[I_{\mathsf{wire}}]$ it holds that*

$$T(\mathsf{pre}_I^C(\mathbf{x}[I_{\mathsf{inp}}], \boldsymbol{\alpha}[I_{\mathsf{wire}}])) = (\mathbf{x}[I_{\mathsf{inp}}], \boldsymbol{\alpha}[I_{\mathsf{wire}}], C).$$

C.1 Proof of Lemma 5.20 (Correctness)

In this section we prove the correctness of the randomized encoding presented in Lemma 5.20. Throughout the end of this subsection, fix a circuit C, an admissible set of gates I, and some I-gate vectors $\mathbf{\Gamma}_I$, and let us denote the output of $T(\mathbf{\Gamma}_I)$ by $(\hat{\mathbf{x}}[I_{\mathsf{inp}}], \hat{\boldsymbol{\alpha}}[I_{\mathsf{wire}}], \hat{C})$. Our goal is to show that,

for every binary $\bar{I}$-input vector $\mathbf{x}[\bar{I}_{\mathsf{inp}}]$ and a binary $\bar{I}$-mask vector $\boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}]$ and every fixing of the randomness $(\mathbf{s}, \mathbf{R})$ of enc, when dec^C is applied to

$$\mathsf{enc}^C\left(\mathbf{\Gamma}_I, \mathsf{pre}_{\bar{I}}^C(\mathbf{x}[\bar{I}_{\mathsf{inp}}], \boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}]); (\mathbf{s}, \mathbf{R})\right)$$

the outcome agrees with the garbled-evaluation function $\mathsf{gEval}_{\hat{C}}$ applied to

$$\left((\hat{\mathbf{x}}[I_{\mathsf{inp}}], \mathbf{x}[\bar{I}_{\mathsf{inp}}]), (\hat{\boldsymbol{\alpha}}[I_{\mathsf{wire}}], \boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}])\right).$$

Fix a binary $\bar{I}$-input vector $\mathbf{x}[\bar{I}_{\mathsf{inp}}]$ and a binary $\bar{I}$-mask vector $\boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}]$, and fix some internal randomness for enc that consists of wire-keys $\mathbf{s} = (s_j^0, s_j^1)_{j\in[m]}$ and randomness $\mathbf{R} = (\mathbf{R}_g)_{g\in C}$ for the gadgets employed for every gate g.

Let $(v_j)_{j\in[m]}$ denote the outputs of the decoder. The following claim shows that the wire-keys and table entries that are computed by the decoder as intermediate values are consistent with corresponding values that are computed by enc.

Claim C.3. *Consider the application of* dec^C *on*

$$\mathsf{enc}^C\left(\mathbf{\Gamma}_I, \mathsf{pre}_{\bar{I}}^C(\mathbf{x}[\bar{I}_{\mathsf{inp}}], \boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}]; (\mathbf{s}, \mathbf{R}))\right),$$

and recall that for each wire j the decoding algorithm dec *computes a pair (v_j, s_j), where $v_j \in \{0,1\}$ and $s_j \in \mathbb{F}^{\omega_j}$. Then for each $j \in [m]$, the key s_j computed by the decoder agrees with the wire-key $s_j^{v_j}$ that was employed by* enc.

Therefore, for each local gate with incoming wires c and d, the value $q_g^{v_c,v_d}$ computed by the decoder agrees agrees with the corresponding value that is computed by enc. *Similarly, for each transmission gate with incoming wire c, the value of $q_g^{v_c}$ as computed by the decoder agrees with the corresponding value as computed by* enc.

Proof. Let $g_1, \ldots, g_N$ be a topological ordering of the gates of C. We prove by induction on $i \in [N]$ that the the property holds for the wire going out from g_i.

Base case. The base case $i = 1$ is an input gate with outgoing wire k. Then, by the perfect correctness of the select gadget the decoder computes $(v_k, s_k^{v_k})$, where $s_k^{v_k}$ is indeed the wire-key whose index is v_k used to construct the gate.

Induction step. If g_i is an input gate then the same proof as in the base case applies here as well. Assume that g_i is a local gate with incoming wires c, d and outgoing wire k. By the induction hypothesis the claim holds for c and d, and so $q_g^{v_c,v_d}$ is correctly recovered by dec. Then, by the perfect correctness of the select gadget the decoder computes $(v_k, s_k^{v_k})$, where $s_k^{v_k}$ is indeed the wire-key whose index is v_k used to construct the gate.

Assume that g_i is a transmission gate with incoming wire c and outgoing wires $k_1, \ldots, k_p$. By the induction hypothesis the claim holds for c, and so $q_g^{v_c}$ is correctly recovered by dec. Fix some $j \in [p]$. Then, by the perfect correctness of the select gadget the decoder computes $(v_{k_j}, s_{k_j}^{v_{k_j}})$, where $s_{k_j}^{v_{k_j}}$ is indeed the wire-key whose index is v_{k_j} used to construct the gate. □

The correctness of the randomized encoding is established in the following lemma.

Lemma C.4 (correctness). *For wire $j \in [m]$, let u_j be the value of wire j in the computation of*

$$\hat{C}(\hat{\mathbf{x}}[I_{\mathsf{inp}}], \mathbf{x}[\bar{I}_{\mathsf{inp}}]),$$

and let v_j be the masked value of wire j, according to

$$\mathsf{dec}^C\left(\mathsf{enc}^C\left(\mathbf{\Gamma}_I, \mathsf{pre}_I^C(\mathbf{x}[\bar{I}_{\mathsf{inp}}], \boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}]); (\mathbf{s}, \mathbf{R})\right)\right).$$

Then for every $j \in I_{\mathsf{wire}}$ it holds that

$$v_j = u_j + \hat{\alpha}_j,$$

and for every $j \in \bar{I}_{\mathsf{wire}}$ it holds that

$$v_j = u_j + \alpha_j.$$

Note that Lemma C.4 implies the correctness of the randomized encoding by the decoder dec, since for wire $j \in [m]$ the output of $\mathsf{gEval}_{\hat{C}}$ is $u_j + \alpha_j$ when $j \in \bar{I}_{\mathsf{wire}}$, and $u_j + \hat{\alpha}_j$ when $j \in I_{\mathsf{wire}}$. We continue with the proof of Lemma C.4.

Proof. Fix some topological ordering of the gates of C, denoted $g_1, \ldots, g_N$. We prove by induction on $i \in [N]$ that if the property holds for the wires going into g_i, then the property holds for the wires going from g_i. Recall that by Claim C.3 for each local gate with incoming wires c and d, the value $q_g^{v_c, v_d}$ is correctly computed by dec, and for each transmission gate with incoming wire c, the value $q_g^{v_c}$ is correctly computed by dec. Since the decoder accesses only those entries, we need to analyse those accesses. First we analyse the case in which $g_i \notin I$.

- Assume that $g_i \notin I$ is an input gate with outgoing wire k. Since $g_i \notin I$ then $k \in \bar{I}_{\mathsf{wire}}$ (see Remark 5.19), so $\alpha_k, x_g \in \{0,1\}$. It follows that $u_k := x_{g_i}$ and since $x_{g_i} + \alpha_k \in \{0,1\}$, then dec sets $v_k := x_{g_i} + \alpha_k = u_k + \alpha_k$, and the claim follows.

- Assume that $g_i \notin I$ is a local gate, with incoming wires c, d and outgoing wire k. Since $g_i \notin I$ then $c, d, k \in \bar{I}_{\mathsf{wire}}$ (see Remark 5.19), so $\alpha_c, \alpha_d, \alpha_k \in \{0,1\}$. By the induction hypothesis it holds that $v_c = u_c + \alpha_c$ and $v_d = u_d + \alpha_d$. In $\hat{C}$ it holds that $u_k := G(u_c, u_d)$. Since $\gamma_g^{v_c, v_d}$ was computed according to pre, we conclude that $\gamma_g^{v_c, v_d} = G(v_c - \alpha_c, v_d - \alpha_d) \in \{0,1\}$, so v_k is defined by dec to be $v_k = \gamma_g^{v_c, v_d} + \alpha_k = G(v_c - \alpha_c, v_d - \alpha_d) + \alpha_k = G(u_c, u_d) + \alpha_k = u_k + \alpha_k$, as required.

- Assume that $g_i \notin I$ is a transmission gates with incoming wire c and outgoing wires $k_1, \ldots, k_p$. Since $g_i \notin I$ then $c \in \bar{I}_{\mathsf{wire}}$ (see Remark 5.19), so $\alpha_c \in \{0,1\}$. By the induction hypothesis it holds that $v_c = u_c + \alpha_c$. Fix some $j \in [p]$ with labels (a, b), so the value of wire k_j in $\hat{C}$ is $u_{k_j} = a \cdot u_c + b$.

  Assume that $k_j \in \bar{I}_{\mathsf{wire}}$, so $\alpha_{k_j} \in \{0,1\}$. In this case dec sets $v_{k_j} := a(v_c - \alpha_c) + b + \alpha_{k_j} = a \cdot u_c + b + \alpha_{k_j} = u_{k_j} + \alpha_{k_j}$, as required.

  Otherwise $k_j \in I_{\mathsf{wire}}$. If $\alpha_{k_j} \in \{0,1\}$, then, as before, $v_{k_j} := u_{k_j} + \alpha_{k_j} = u_{k_j} + \hat{\alpha}_{k_j}$. Otherwise $\alpha_{k_j} \notin \{0,1\}$, and so the decoding algorithm sets $v_{k_j} := a(v_c - \alpha_c) + b = a \cdot u_c + b = u_{k_j} = u_{k_j} + \hat{\alpha}_{k_j}$, as required.

We continue with the case in which $g_i \in I$.

- Assume that $g_i \in I$ is an input gate with outgoing wire $k \in I_{\mathsf{wire}}$, $\mathbf{\Gamma}_{g_i} = x_{g_i}$ and wire mask α_k. Recall that according to dec, if $x_{g_i} + \alpha_k \in \{0,1\}$ then $v_k := x_{g_i} + \alpha_k$, and otherwise, if $x_{g_i} + \alpha_k \notin \{0,1\}$, then $v_k := 0$. By the definition of T, if $x_{g_i} + \alpha_k \in \{0,1\}$, then $u_k = \hat{x}_{g_i} = (x_{g_i} + \alpha_k) - \hat{\alpha}_k = v_k - \hat{\alpha}_k$, and otherwise $u_k = \hat{x}_{g_i} = 0 - \hat{\alpha}_k = v_k - \hat{\alpha}_k$, as required.

- Assume that $g_i \in I$ is a local gate, with incoming wires c, d and outgoing wire k, where $c, d, k \in I_{\mathsf{wire}}$. By the induction hypothesis it holds that $v_c = u_c + \hat{\alpha}_c$ and $v_d = u_d + \hat{\alpha}_d$. Recall that according to dec, if $\gamma_{g_i}^{v_c,v_d} + \alpha_k \in \{0,1\}$ then $v_k := \gamma_{g_i}^{v_c,v_d} + \alpha_k$, and otherwise, if $\gamma_{g_i}^{v_c,v_d} + \alpha_k \notin \{0,1\}$, then $v_k := 0$. By the definition of T, if $\gamma_{g_i}^{v_c,v_d} + \alpha_k \in \{0,1\}$ then $u_k = \hat{G}(u_c, u_d) = (\gamma_{g_i}^{u_c+\hat{\alpha}_c, u_d+\hat{\alpha}_d} + \alpha_k) - \hat{\alpha}_k = (\gamma_{g_i}^{v_c,v_d} + \alpha_k) - \hat{\alpha}_k = v_k - \hat{\alpha}_k$, and if $\gamma_{g_i}^{v_c,v_d} + \alpha_k \notin \{0,1\}$ then $u_k = 0 - \hat{\alpha}_k = v_k - \hat{\alpha}_k$, as required.

- Assume that $g_i \in I$ is a transmission gates with incoming wire $c \in I_{\mathsf{wire}}$ and outgoing wires $k_1, \ldots, k_p$. By the induction hypothesis it holds that $v_c = u_c + \hat{\alpha}_c$. Fix some wire $j \in [p]$ whose original labels are (a, b).

  We begin with the case $a = 0$, in which, by the definition of T, the effective labels are also (a, b), so $u_{k_j} = 0 \cdot u_c + b = b$. Note that according to dec, if $b + \alpha_{k_j} \in \{0,1\}$ then $v_{k_j} := b + \alpha_{k_j}$, and otherwise, if $b + \alpha_{k_j} \notin \{0,1\}$, then $v_{k_j} := b$. First, assume that k_j is an honest wire (that is, $k_j \in \bar{I}_{\mathsf{wire}}$). In this case it always holds that $\alpha_{k_j} \in \{0,1\}$, and so $v_{k_j} := b + \alpha_{k_j} = u_{k_j} + \alpha_{k_j}$, as required. Assume that $k_j \in I_{\mathsf{wire}}$ is a malicious wire. As before, if $\alpha_{k_j} \in \{0,1\}$ then $v_{k_j} := b + \alpha_{k_j} = u_{k_j} + \alpha_{k_j} = u_{k_j} + \hat{\alpha}_{k_j}$, as required. Otherwise, if $\alpha_{k_j} \notin \{0,1\}$, then $v_{k_j} := b = u_{k_j} = u_{k_j} + \hat{\alpha}_{k_j}$, as required.

  Now, consider the case $a = 1$. Recall that, according to dec, (1) if $\alpha_{k_j} - \alpha_c \in \{0,1\}$ then $v_{k_j} := v_c - \alpha_c + b + \alpha_{k_j}$; (2) if $\alpha_c \notin \{0,1\}$ and $\alpha_{k_j} \in \{0,1\}$ then $v_{k_j} := \alpha_{k_j}$; (3) if $\alpha_c \in \{0,1\}$ and $\alpha_{k_j} \notin \{0,1\}$ then $v_{k_j} := v_c - \alpha_c + b$; (4) if $\alpha_c, \alpha_{k_j} \notin \{0,1\}$ and $\alpha_{k_j} - \alpha_c \notin \{0,1\}$ then $v_{k_j} := 0$.

  Assume that k_j is an honest wire (that is, $k_j \in \bar{I}_{\mathsf{wire}}$), so it holds that $\alpha_{k_j} \in \{0,1\}$. If $\alpha_c \in \{0,1\}$ then (according to case (1)) $v_{k_j} := v_c - \alpha_c + b + \alpha_{k_j} = u_c + \hat{\alpha}_c - \alpha_c + b + \alpha_{k_j} = u_c + b + \alpha_{k_j}$. Furthermore, according to T it holds that the effective labels are (a, b), so $u_{k_j} = u_c + b$ and so $v_{k_j} = u_{k_j} + \alpha_{k_j}$, as required. If $\alpha_c \notin \{0,1\}$, then (according to case (2)) $v_{k_j} = \alpha_{k_j}$, and $u_{k_j} = 0 \cdot u_c + 0 = 0$ (since the effective labels are $(0,0)$), so $v_{k_j} = u_c + \alpha_{k_j}$, as required.

  Assume that $k_j \in I_{\mathsf{wire}}$ is a malicious wire. In case (1), according to T we have that $u_{k_j} = u_c + \hat{\alpha}_c - \alpha_c + b + \alpha_{k_j} - \hat{\alpha}_{k_j}$, and according to the decoding algorithm we have $v_{k_j} = v_c - \alpha_c + b + \alpha_{k_j} = u_c + \hat{\alpha}_c - \alpha_c + b + \alpha_{k_j} = u_{k_j} + \hat{\alpha}_{k_j}$, as required. In case (2), according to T we have $u_{k_j} = 0 \cdot u_c + 0 = 0$ and according to dec we have $v_{k_j} = \alpha_{k_j} = 0 + \hat{\alpha}_{k_j} = u_{k_j} + \hat{\alpha}_{k_j}$, as required. In case (3), according to T we have $u_{k_j} = u_c + b$, and according to dec we have $v_{k_j} = v_c - \alpha_c + b = u_c + \hat{\alpha}_c - \alpha_c + b = u_c + b = u_{k_j} = u_{k_j} + \hat{\alpha}_{k_j}$, as required. In case (4), according to T we have $u_{k_j} = 0$ and according to dec we have $v_{k_j} = 0 = u_{k_j} = u_{k_j} + \hat{\alpha}_{k_j}$, as required.

This concludes the proof. □

C.2 Proof of Lemma 5.20 (Privacy)

Fix a circuit C, an admissible set of gates I, and some I-gate vectors $\mathbf{\Gamma}_I$, and let us denote the output of $T(\mathbf{\Gamma}_I)$ by $(\hat{\mathbf{x}}[I_{\mathsf{inp}}], \hat{\boldsymbol{\alpha}}[I_{\mathsf{wire}}], \hat{C})$. Our goal is to describe a simulator that, for every binary

$\bar{I}$-input vector $\mathbf{x}[\bar{I}_{\mathsf{inp}}]$ and a binary $\bar{I}$-mask vector $\boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}]$, samples the distribution

$$\mathsf{enc}^C\left(\mathbf{\Gamma}_I, \mathsf{pre}_I^C(\mathbf{x}[\bar{I}_{\mathsf{inp}}], \boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}])\right),$$

(induced by the internal randomness of enc^C) given the output of

$$\mathsf{gEval}_{\hat{C}}\left((\hat{\mathbf{x}}[I_{\mathsf{inp}}], \mathbf{x}[\bar{I}_{\mathsf{inp}}]), (\hat{\boldsymbol{\alpha}}[I_{\mathsf{wire}}], \boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}])\right).$$

Specifically, the simulator takes a binary vector $(v_j)_{j\in[m]}$ as input, and outputs a list of gate tables $(\widetilde{Q}_g)_{g\in C}$. Recall that the simulator takes the set I and an I-gate vector $\mathbf{\Gamma}_I$ as an auxiliary input. We begin with the description of the simulator.

The simulator $\mathcal{S}$. The simulator first samples wire-keys $\{\tilde{s}_j^{v_j}\}_{j\in[m]}$ uniformly at random, where $\tilde{s}_j^{v_j} \in \mathbb{F}^{\omega_j}$. We call those keys the *on-path* keys, since those are the keys that will be revealed in the computation of dec on the gate tables that we construct. The *on-path* keys will be used to compute, in each gate g, a single *on-path* entry of the gate's table, whereas all the *off-path* entries of the gate will be sampled uniformly at random. Here the on-path entry of the gate is the entry that is revealed to the decoder; That is, in an input gate g the only entry $\widetilde{Q}_g$ is the on-path entry, in a local computation gate with incoming wires c and d the on-path entry of g is $\widetilde{Q}_g^{v_c,v_d}$, and in a generalized transmission gate with with a single incoming wire c, the on-path entry is $\widetilde{Q}_g^{v_c}$. Details follow.

For every gate $g \in C$ the simulator samples the gate table $\widetilde{Q}_g$ according to the gate's type as follows.

- **g is an input gate with outgoing wire k.** If $g \notin I$, let $\widetilde{Q}_g$ be the output of the simulator of select when applied on $(v_k, \tilde{s}_k^{v_k})$. If $g \in I$, let $\widetilde{Q}_g$ be the output of the simulator of select when applied on $(x_g + \alpha_k, \tilde{s}_k^{v_k})$. (In this case g is a malicious gate, so Q_g is constructed with the value $x_g + \alpha_k$, taken from $\mathbf{\Gamma}_I$.)

- **g is a local computation gate with incoming wires c, d and outgoing wire k.** If $g \notin I$, let $\widetilde{q}_g^{v_c,v_d}$ be the output of the simulator of select when applied on $(v_k, \tilde{s}_k^{v_k})$. If $g \in I$, let $\widetilde{q}_g^{v_c,v_d}$ be the output of the simulator of select when applied on $(\gamma_g^{v_c,v_d} + \alpha_k, \tilde{s}_k^{v_k})$. (In this case g is a malicious gate, so $q_g^{v_c,v_d}$ is constructed with the value $\gamma_g^{v_c,v_d} + \alpha_k$ taken from $\mathbf{\Gamma}_I$.) Let $\widetilde{Q}_g^{v_c,v_d} := \widetilde{q}_g^{v_c,v_d} + \tilde{s}^{v_c}[v_d] + \tilde{s}^{v_d}[v_c]$. For $(\beta_c, \beta_d) \neq (v_c, v_d)$, sample $\widetilde{Q}_g^{\beta_c,\beta_d}$ uniformly at random. Let $\widetilde{Q}_g := (\widetilde{Q}_g^{0,0}, \widetilde{Q}_g^{0,1}, \widetilde{Q}_g^{1,0}, \widetilde{Q}_g^{1,1})$.

- **g is a transmission gate with incoming wire c, and outgoing wires $k_1, \ldots, k_p$.** Denote the original labels of wire k_j, as defined by C, by (a_j, b_j). Recall that the gate table of a transmission gate is constructed using the $\mathsf{tselect}$ gadget, and that the g_{tselect} function reveals all the selectors up to the first binary selector (if such exists), and the key that corresponds to the effective selector. Therefore, for each wire $j \in [m]$ we first identify how the output of the corresponding g_{tselect} should look like by identifying the first binary selector, and then apply the simulator of $\mathsf{tselect}$ on this value.

  If $g \notin I$, for every $j \in [m]$ we split into cases.

  - Assume that $k_j \in \bar{I}_{\mathsf{wire}}$. In this case let $\widetilde{q}_g^{v_c}[j]$ be the output of the simulator of $\mathsf{tselect}$ when applied on $(v_{k_j}, \tilde{s}_{k_j}^{v_{k_j}})$.

– Assume that $k_j \in I_{\mathsf{wire}}$ and $\alpha_{k_j} \in \{0,1\}$. In this case let $\widetilde{q}_g^{v_c}[j]$ be the output of the simulator of $\mathsf{tselect}$ when applied on $(v_{k_j}, \tilde{s}_{k_j}^{v_{k_j}})$.

– Assume that $k_j \in I_{\mathsf{wire}}$ and $\alpha_{k_j} \notin \{0,1\}$. In this case let $\widetilde{q}_g^{v_c}[j]$ be the output of the simulator of $\mathsf{tselect}$ when applied on $(v_{k_j} + \alpha_{k_j}, \alpha_{k_j}, v_{k_j}, \tilde{s}_{k_j}^{v_{k_j}})$.

If $g \in I$, for every $j \in [m]$ we split into cases.

– Assume that $k_j \in I_{\mathsf{wire}}$ and $a_j \cdot (v_c - \alpha_c) + b_j + \alpha_{k_j} \in \{0,1\}$. In this case, let $\widetilde{q}_g^{v_c}[j]$ be the output of the simulator of $\mathsf{tselect}$ when applied on $(v_{k_j}, \tilde{s}_{k_j}^{v_{k_j}})$.

– Assume that $k_j \in \bar{I}_{\mathsf{wire}}$ and $a_j \cdot (v_c - \alpha_c) + b_j \in \{0,1\}$. In this case, let $\widetilde{q}_g^{v_c}[j]$ be the output of the simulator of $\mathsf{tselect}$ when applied on $(v_{k_j}, \tilde{s}_{k_j}^{v_{k_j}})$.

– Assume that $k_j \in I_{\mathsf{wire}}$, $a_j \cdot (v_c - \alpha_c) + b_j + \alpha_{k_j} \notin \{0,1\}$ and $\alpha_{k_j} \in \{0,1\}$. In this case, In this case, let $\widetilde{q}_g^{v_c}[j]$ be the output of the simulator of $\mathsf{tselect}$ when applied on $(a_j \cdot (v_c - \alpha_c) + b_j + v_{k_j}, v_{k_j}, \tilde{s}_{k_j}^{v_{k_j}})$.

– Assume that $k_j \in \bar{I}_{\mathsf{wire}}$, and $a_j \cdot (v_c - \alpha_c) + b_j \notin \{0,1\}$. In this case, let $\widetilde{q}_g^{v_c}[j]$ be the output of the simulator of $\mathsf{tselect}$ when applied on $(a_j \cdot (v_c - \alpha_c) + b_j + v_{k_j}, v_{k_j}, \tilde{s}_{k_j}^{v_{k_j}})$.

– Assume that $k_j \in I_{\mathsf{wire}}$, $a_j(v_c - \alpha_c) + b_j + \alpha_{k_j} \notin \{0,1\}$, $\alpha_{k_j} \notin \{0,1\}$ and $a_j(v_c - \alpha_c) + b_j \in \{0,1\}$. In this case let let $\widetilde{q}_g^{v_c}[j]$ be the output of the simulator of $\mathsf{tselect}$ when applied on $(a_j \cdot (v_c - \alpha_c) + b_j + \alpha_{k_j}, \alpha_{k_j}, v_{k_j}, \tilde{s}_{k_j}^{v_{k_j}})$.

– Assume that $k_j \in I_{\mathsf{wire}}$, $a_j(v_c - \alpha_c) + b_j + \alpha_{k_j} \notin \{0,1\}$, $\alpha_{k_j} \notin \{0,1\}$ and $a_j(v_c - \alpha_c) + b_j \notin \{0,1\}$. In this case let let $\widetilde{q}_g^{v_c}[j]$ be the output of the simulator of $\mathsf{tselect}$ when applied on $(a_j \cdot (v_c - \alpha_c) + b_j + \alpha_{k_j}, \alpha_{k_j}, a_j \cdot (v_c - \alpha_c) + b_j, \tilde{s}_{k_j}^{v_{k_j}})$.

Let $\widetilde{q}_g^{v_c} := (\widetilde{q}_g^{v_c}[1], \ldots, \widetilde{q}_g^{v_c}[p])$, and $\widetilde{Q}_g^{v_c} := \widetilde{q}_g^{v_c} + \tilde{s}_c^{v_c}$. Let $\widetilde{Q}_g := (\widetilde{Q}_g^0, \widetilde{Q}_g^1)$, where $\widetilde{Q}_g^{1-v_c}$ is sampled uniformly at random.

Finally, the output of the simulator is $(\widetilde{Q}_g)_{g \in C}$.

Privacy. Fix some binary vectors $\mathbf{x}[\bar{I}_{\mathsf{inp}}]$ and $\boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}]$ and let $\mathbf{v} = (v_j)_{j \in [m]}$ denote the output of $\mathsf{gEval}_{\hat{C}}\left((\hat{\mathbf{x}}[I_{\mathsf{inp}}], \mathbf{x}[\bar{I}_{\mathsf{inp}}]), (\hat{\boldsymbol{\alpha}}[I_{\mathsf{wire}}], \boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}])\right)$. We need to show that the distribution of "simulated" gate tables $(\widetilde{Q}_g)_{g \in C}$, as defined by

$$\mathcal{S}(\mathbf{v}),$$

is the same as the distribution of the "real" gate tables $(Q_g)_{g \in C}$, which are sampled from

$$\mathsf{enc}^C\left(\mathbf{\Gamma}_I, \mathsf{pre}_I^C(\mathbf{x}[\bar{I}_{\mathsf{inp}}], \boldsymbol{\alpha}[\bar{I}_{\mathsf{wire}}])\right).$$

Alignment of on-path entries and on-path keys. Recall that any fixed vector of gate tables $\mathcal{Q} = (\mathcal{Q}_g)$ induces a sequence of garbled wire values (obtained by invoking the decoder dec) which in turn can be used to define the on-path and off-path entries of the tables. Furthermore, by the design, the outcome of the simulator $(\widetilde{Q}_g)_{g \in C}$ always satisfies $\mathsf{dec}((\widetilde{Q}_g)_{g \in C}) = \mathbf{v}$. Also, by correctness (Lemma C.4), the same holds for the real gate tables, i.e., $\mathsf{dec}((Q_g)_{g \in C}) = \mathbf{v}$. Hence, the on-path entries of the gate tables in both experiments are aligned with each other.

From now on, let us condition on the event that both the simulated on-path keys $(\tilde{s}_j^{v_j})_{j\in[m]}$ and the real on-path keys $(s_j^{v_j})_{j\in[m]}$ agree with each other and are fixed to some arbitrary values. It suffices to show that, conditioned on this event, the simulated gate tables, $(\widetilde{Q}_g)_{g\in C}$, and real gate tables, $(Q_g)_{g\in C}$, are distributed identically.

Let $g_1,\ldots,g_N$ be a topological ordering of the gates in C. Before establishing privacy, we show that the off-path entries in "real" gate tables are uniformly distributed. Specifically, this will follow from the following claim.

Claim C.5. *For every fixed gate number $i \in [N]$ and every fixing of the off-path entries of $(Q_{g_1},\ldots,Q_{g_{i-1}})$, the conditional joint distribution of the off-path entries of the i-th gate together with all the off-path keys of wires that do not enter any of the first i gates is uniform.*

Proof of Claim C.5. Fix the internal randomness of the select and tselect gadgets which are used in off-path entries of the gate tables. We show that the claim holds for every such fixing.

For an index i let us denote by W_i the set of all wires that do not enter any of the first i gates. We prove the claim by induction on i.

The basis $i = 1$ holds since the first gate must be an input gate and its gate table outputs the on-path key (and the masked value of the wire) and is therefore independent of the off-path keys of all wires. (The off-path entry of this gate is empty and so we can say that it is distributed uniformly over the set of empty strings.)

Let us assume that the claim holds for i, and prove it for $i+1$. The hypothesis implies that the vector $(s_w^{1-v_w})_{w\in W_i}$ is uniformly distributed even when conditioned on any fixing of the off-path entries of $(Q_{g_1},\ldots,Q_{g_{i-1}})$ and on any fixing of the off-path entries of Q_{g_i}. Our goal is to show that the joint distribution of (1) the off-path entries of $Q_{g_{i+1}}$ and (2) the W_{i+1}-off-path keys $(s_w^{1-v_w})_{w\in W_{i+1}}$, is uniform. Since $W_{i+1}\subset W_i$, our hypothesis already implies that (2) holds. It therefore suffices to show that even when we condition on any fixing of $(s_w^{1-v_w})_{w\in W_{i+1}}$ the off-path entries of $Q_{g_{i+1}}$ are uniformly distributed.

If $g = g_{i+1}$ is an input gate, this trivially holds since the off-path entry of $Q_{g_{i+1}}$ is empty. Next, assume that g is a local computation gate with input wires c and d and an output wire w and observe that $c,d \in W_i \setminus W_{i+1}$ and that $w \in W_{i+1}$ and therefore the w-th off-path key is fixed. We show that the off-path entries $Q_g^{\beta_c,\beta_d}$ for $(\beta_c,\beta_d)\neq(v_c,v_d)$ are uniformly distributed even when all the randomness in the experiment is fixed except for the off-path keys of the wires c and d. Indeed, recall that

$$Q_g^{\beta_c,\beta_d} = q_g^{\beta_c,\beta_d} + \left(s_c^{\beta_c}[\beta_d] + s_d^{\beta_d}[\beta_c]\right),$$

where $q_g^{\beta_c,\beta_d}$ is some fixed value (that depends on the keys of the wire w and other randomness that was already fixed). The claim follows by noting that every off-path entry $Q_g^{\beta_c,\beta_d}$ where $(\beta_c,\beta_d)\neq(v_c,v_d)$, is randomized by a different half of the random off-path keys $s_c^{1-v_c}, s_d^{1-v_d}$ of the incoming wires. Specifically, $Q^{v_c,1-v_d}$ is randomized by $s_d^{1-v_d}[v_c]$, the entry Q^{1-v_c,v_d} is randomized by $s_c^{1-v_c}[v_d]$ and the entry $Q^{1-v_c,1-v_d}$ is randomized by $s_c^{1-v_c}[1-v_d]$.

The case of generalized transmission gates is similar. Assume that g is a generalized transmission gate with input wire c, and outgoing wires $k_1,\ldots,k_p$, and observe that $c\in W_i\setminus W_{i+1}$ and $k_1,\ldots,k_p\in W_{i+1}$, and so the off-path keys of $k_1,\ldots,k_p$ are fixed. We show that the off-path entry $Q_g^{1-v_c}$ is uniformly distributed even when all the randomness in the experiment is fixed except for the off-path keys of wire c. Indeed, since

$$Q_g^{1-v_c} = q_g^{1-v_c} + s_c^{1-v_c},$$

where $q_g^{1-v_c}$ is some fixed value (that depends on the keys of the wires $k_1, \ldots, k_p$ and other randomness that was already fixed), and $s_c^{1-v_c}$ is uniformly distributed, the claim follows.

□

Recall that the simulator samples the off-path entries of the gate tables uniformly at random. The above claim therefore shows that the marginal distribution of the off-path entries in both experiments (real and simulated) is uniform.

Let us further condition on the event that the off-path entries in both experiments are fixed to the same arbitrary value. To complete the privacy proof, it suffices to show that the on-path entries in both experiments are distributed identically. This will follow from the following claim.

Claim C.6. *For every $i \in [N]$, conditioned on the event that the on-path entries of $(Q_{g_1}, \ldots, Q_{g_{i-1}})$ and $(\widetilde{Q}_{g_1}, \ldots, \widetilde{Q}_{g_{i-1}})$ take the same fixed value, the on-path entry of Q_{g_i} and the on-path entry of $\widetilde{Q}_{g_i}$ are identically distributed.*

Proof. The proof is by induction on i. Let $g = g_i$. Since the off-path entries of the gates and the on-path keys are all fixed, the only randomness involved is the internal randomness of the gadget (in the real experiment) and the randomness of the gadget's simulator (in the simulated experiments). Hence, by the privacy of the gadget's simulators, it suffices to show that whenever select (resp., $\mathsf{tselect}$) is applied in the real encoding to inputs (γ, s^0, s^1) (resp., $(\gamma_1, \gamma_2, \gamma_3, s^0, s^1)$) the corresponding simulator is applied on $g_{\mathsf{select}}(\gamma, s^0, s^1)$ (resp., $g_{\mathsf{tselect}}(\gamma_1, \gamma_2, \gamma_3, s^0, s^1)$). It will be also useful to keep in mind the following fact: Recall that we can associate a single gadget with every wire j that leaves the gate g; then, by the correctness analysis (Lemma C.4), the effective selector in the gadget always equal to the garbled value v_j of the corresponding wire. We continue the proof via case analysis.

g is an input gate with outgoing wire k. The on-path entry Q_g in the real experiment is $Q_g = \mathsf{select}(x_g + \alpha_k, s_k^0, s_k^1)$. If $g \notin I$ then $v_k = x_g + \alpha_k$ is binary and so $g_{\mathsf{select}}(x_g + \alpha_k, s_k^0, s_k^1)$ outputs $(v_k, s_k^{v_k})$. Indeed, the simulated table $\widetilde{Q}_g$ is sampled by applying the select-simulator to $(v_k, s_k^{v_k})$, as required. Otherwise, if $g \in I$, then $g_{\mathsf{select}}(x_g + \alpha_k, s_k^0, s_k^1)$ outputs $x_g + \alpha_k$ together with the key s_k^b where the effective selector b equals to v_k by the correctness of the garbled circuit (Lemma C.4). Since the simulated table $\widetilde{Q}_g$ is sampled by applying the select-simulator to $(x_g + \alpha_k, s_k^{v_k})$, the privacy of select guarantees that Q_g and $\widetilde{Q}_g$ are identically distributed. (Recall that $x_g + \alpha_k$ is extracted from $\mathbf{\Gamma}_I$.)

g is a local gate with incoming wires c, d and outgoing wire k. Recall that $Q_g^{v_c, v_d}$ and $\widetilde{Q}_g^{v_c, v_d}$ are obtained by encrypting the values $q_g^{v_c, v_d}$ and $\widetilde{q}_g^{v_c, v_d}$ under the same on-path keys $(s_c^{v_c}, s_d^{v_d})$. Therefore it suffices to show that $q_g^{v_c, v_d}$ and $\widetilde{q}_g^{v_c, v_d}$ are identically distributed. In the real experiment,

$$q_g^{v_c, v_d} = \mathsf{select}(\gamma_g^{v_c, v_d} + \alpha_k, s_k^0, s_k^1).$$

If $g \notin I$, then $v_k = \gamma_g^{v_c, v_d} + \alpha_k$ is binary and so $g_{\mathsf{select}}(\gamma_g^{v_c, v_d} + \alpha_k, s_k^0, s_k^1) = (v_k, s_k^{v_k})$. Since the simulated entry $\widetilde{q}_g^{v_c, v_d}$ is sampled by applying the select-simulator to $(v_k, s_k^{v_k})$, the resulting distribution is identical to $q_g^{v_c, v_d}$.

If $g \in I$ then $g_{\mathsf{select}}(\gamma_g^{v_c, v_d} + \alpha_k, s_k^0, s_k^1) = (\gamma_g^{v_c, v_d} + \alpha_k, s_k^b)$ where the effective selector $b = v_k$ as follows from the correctness analysis (Lemma C.4). Since the simulated entry $\widetilde{q}_g^{v_c, v_d}$ is sampled by applying the select-simulator to $(\gamma_g^{v_c, v_d} + \alpha_k, s_k^{v_k})$, the perfect privacy of select implies that $q_g^{v_c, v_d}$ and

$\widetilde{q}_g^{v_c,v_d}$ are identically distributed. (Again, recall that the simulator retrieves the value $\gamma_g^{v_c,v_d} + \alpha_k$ from $\mathbf{\Gamma}_I$.)

g is a generalized transmission gate with incoming wire d and outgoing wires $k_1, \ldots, k_p$. As in the previous case, it suffices to show that $q_g^{v_c}$ has the same distribution as $\widetilde{q}_g^{v_c}$. Since $\{q_g^{v_c}[j]\}_{j\in[p]}$ (resp. $\{\widetilde{q}_g^{v_c}[j]\}_{j\in[p]}$) are independent random variables, it is enough to show that for every $j \in [p]$ it holds that $q_g^{v_c}[j]$ has the same distribution as $\widetilde{q}_g^{v_c}[j]$. Fix some $j \in [p]$ with label (a_j, b_j) and recall that enc samples

$$q_g^{v_c}[j] = \mathsf{tselect}(\delta_j + \alpha_{k_j}, \alpha_{k_j}, \delta_j, s_{k_j}^0, s_{k_j}^1)$$

where $\delta_j = a_j \cdot (v_c - \alpha_c) + b_j$.

- Assume that $g \notin I$. Hence, $c \in \bar{I}_{\mathsf{wire}}$, $\alpha_c \in \{0,1\}$ and δ_j is also binary. We split into cases.
  - If $k_j \in \bar{I}_{\mathsf{wire}}$, then $\alpha_{k_j} \in \{0,1\}$. Therefore, the first selector $\delta_j + \alpha_{k_j}$ is binary and $g_{\mathsf{tselect}}(\delta_j + \alpha_{k_j}, \alpha_{k_j}, \delta_j, s_{k_j}^0, s_{k_j}^1)$ outputs $\delta_j + \alpha_{k_j}$ together with the corresponding key $s_{k_j}^{\delta_j+\alpha_{k_j}}$. By the correctness of the encoding (Lemma C.4), it holds that the effective selector $\delta_j + \alpha_{k_j}$ equals to v_{k_j}, and we conclude that g_{tselect} outputs $(v_{k_j}, s_{k_j}^{v_{k_j}})$. Since the simulator samples $\widetilde{q}_g^{v_c}[j]$ by applying the $\mathsf{tselect}$-simulator to $(v_{k_j}, s_{k_j}^{v_{k_j}})$, the claim follows from the perfect privacy of $\mathsf{tselect}$.
  - If $k_j \in I_{\mathsf{wire}}$ and $\alpha_{k_j} \in \{0,1\}$. (Recall that α_{k_j} is extracted from $\mathbf{\Gamma}_I$). Then, as in the previous case, g_{tselect} outputs $(v_{k_j}, s_{k_j}^{v_{k_j}})$, and since $\widetilde{q}_g^{v_c}[j]$ is sampled by applying the $\mathsf{tselect}$-simulator to $(v_{k_j}, s_{k_j}^{v_{k_j}})$, the simulated distribution is identical to the real one.
  - If $k_j \in I_{\mathsf{wire}}$ and α_{k_j} (extracted from $\mathbf{\Gamma}_I$) is non-binary, then only the third selector, δ_j, of the $\mathsf{tselect}$ gadget is binary and first two selectors, $\delta_j + \alpha_{k_j}$ and α_{k_j} are non-binary. Therefore, $g_{\mathsf{tselect}}(\delta_j + \alpha_{k_j}, \alpha_{k_j}, \delta_j, s_{k_j}^0, s_{k_j}^1)$ outputs all three selectors together with the key $s_{k_j}^{\delta_j}$. By the correctness of the encoding (Lemma C.4), the effective selector δ_j equals to v_{k_j}. Since $\widetilde{q}_g^{v_c}[j]$ is indeed sampled by applying the $\mathsf{tselect}$-simulator on $(v_{k_j} + \alpha_{k_j}, \alpha_{k_j}, v_{k_j}, s_{k_j}^{v_{k_j}})$, the claim follows from the perfect privacy of $\mathsf{tselect}$.
- Assume that $g \in I$, and so α_c is known given $\mathbf{\Gamma}_I$. We split into cases.
  - If $k_j \in I_{\mathsf{wire}}$ (so α_{k_j} is known given $\mathbf{\Gamma}_I$) and the first selector $\delta_j + \alpha_{k_j}$ is binary, then g_{tselect} treats this value as the effective selector and outputs it together with the corresponding s_{k_j} key. By the correctness of the encoding (Lemma C.4), the effective selector equals to v_{k_j}, and so the output of g_{tselect} is the pair $(v_{k_j}, s_{k_j}^{v_{k_j}})$. Since we sample $\widetilde{q}_g^{v_c}[j]$ by applying the $\mathsf{tselect}$-simulator on $(v_{k_j}, s_{k_j}^{v_{k_j}})$, the claim follows from the perfect privacy of $\mathsf{tselect}$.
  - If $k_j \in \bar{I}_{\mathsf{wire}}$ (so that $\alpha_{k_j} \in \{0,1\}$) and, in addition, δ_j is binary, then the situation is identical to the previous case. That is, the first selector $\delta_j + \alpha_{k_j}$ is binary, and it plays the role of the effective selector v_{k_j}. The output of g_{tselect} is the pair $(v_{k_j}, s_{k_j}^{v_{k_j}})$, and $\widetilde{q}_g^{v_c}[j]$ is distributed just like $q_g^{v_c}[j]$ since it is sampled by applying the $\mathsf{tselect}$-simulator on $(v_{k_j}, s_{k_j}^{v_{k_j}})$.

– If $k_j \in I_{\mathsf{wire}}$ (so α_{k_j} is known given $\mathbf{\Gamma}_I$), the first selector $\delta_j + \alpha_{k_j}$ is non-binary but the second selector α_{k_j} is binary, then g_{tselect} uses the second selector as its effective selector, i.e., $v_{k_j} = \alpha_{k_j}$ and outputs the tuple $z = (\delta_j + \alpha_{k_j}, v_{k_j}, s_{k_j}^{v_{k_j}})$. Since we compute $\widetilde{q}_g^{v_c}[j]$ by applying the $\mathsf{tselect}$-simulator on z, the claim follows from the perfect privacy of $\mathsf{tselect}$.

– If $k_j \in \bar{I}_{\mathsf{wire}}$ (so $\alpha_{k_j} \in \{0,1\}$), and δ_j is non-binary, the situation is similar to the previous case. The first selector $\delta_j + \alpha_{k_j}$is non-binary, but the second selector, $\alpha_{k_j}$is binary, so $v_{k_j} = \alpha_{k_j}$, and g_{tselect} outputs the tuple $z = (\delta_j + \alpha_{k_j}, v_{k_j}, s_{k_j}^{v_{k_j}})$. Since $\widetilde{q}_g^{v_c}[j]$ is sampled by applying the $\mathsf{tselect}$-simulator on z it is distributed identically to $q_g^{v_c}[j]$.

– If $k_j \in I_{\mathsf{wire}}$ (so α_{k_j} is known given $\mathbf{\Gamma}_I$), the first selector $\delta_j + \alpha_{k_j}$ is non-binary, the second selector α_{k_j} is non-binary and the third selector δ_j is binary, then g_{tselect} uses δ_j as its effective selector v_{k_j} and outputs the tuple $z = (\delta_j + \alpha_{k_j}, \alpha_{k_j}, v_{k_j}, s_{k_j}^{v_{k_j}})$. Since $\widetilde{q}_g^{v_c}[j]$ is sampled by applying the $\mathsf{tselect}$-simulator on z it is distributed identically to $q_g^{v_c}[j]$.

– If $k_j \in I_{\mathsf{wire}}$ (so α_{k_j} is known given $\mathbf{\Gamma}_I$), and all three selectors, $\delta_j + \alpha_{k_j}$, α_{k_j} and δ_j are non-binary, then g_{tselect} uses 0 as its effective selector v_{k_j}, and outputs the tuple $z = (\delta_j + \alpha_{k_j}, \alpha_{k_j}, \delta_j, s_{k_j}^{v_{k_j}})$. Since $\widetilde{q}_g^{v_c}[j]$ is sampled by applying the $\mathsf{tselect}$-simulator on z, it is distributed identically to $q_g^{v_c}[j]$.

This completes the proof of the claim, and completes the privacy analysis. □

ECCC ISSN 1433-8092
https://eccc.weizmann.ac.il