A Full Proof of the BGW Protocol for Perfectly-Secure Multiparty Computation[*]

Gilad Asharov[†] Yehuda Lindell[†]

July 8, 2011

Abstract

In the setting of secure multiparty computation, a set of n parties with private inputs wish to jointly compute some functionality of their inputs. One of the most fundamental results of information-theoretically secure computation was presented by Ben-Or, Goldwasser and Wigderson (BGW) in 1988. They demonstrated that any n-party functionality can be computed with *perfect security*, in the private channels model. When the adversary is semi-honest this holds as long as $t < n/2$ parties are corrupted, and when the adversary is malicious this holds as long as $t < n/3$ parties are corrupted. Unfortunately, a full detailed proof of these results was never given. In this paper, we remedy this situation and provide a full proof of security of the BGW protocol. We also derive corollaries for security in the presence of adaptive adversaries and under concurrent general composition (equivalently, universal composability). In addition, we give a full specification of the protocol for the malicious setting. This includes one new step for the perfect multiplication protocol in the case of $n/4 \le t < n/3$.

[*]This research was generously supported by the European Research Council as part of the ERC project LAST.
[†]Dept. of Computer Science, Bar-Ilan University, Israel. email: `asharog@cs.biu.ac.il, lindell@biu.ac.il.`

ISSN 1433-8092

Contents

1 Introduction

1.1 Background – Secure Computation

In the setting of secure multiparty computation, a set of n parties with possibly private inputs wish to securely compute some function of their inputs in the presence of adversarial behavior. Loosely speaking, the security requirements from such a computation are that nothing is learned from the protocol other than the output (*privacy*), that the output is distributed according to the prescribed functionality (*correctness*), that parties cannot choose their inputs as a function of the others' inputs (*independence of inputs*), and that all parties receive output (*fairness* and *guaranteed output delivery*). The actual definition [19, 24, 3, 5] formalizes this by comparing the result of a real protocol execution with the result of an ideal execution in an ideal model where an incorruptible trusted party carries out the computation for the parties. This definition has come to be known as the "ideal/real simulation paradigm".

There are many different settings within which secure computation has been considered. Regarding the adversary, one can consider semi-honest adversaries (who follow the protocol specification but try to learn more than they should by inspecting the protocol transcript) or malicious adversaries (who follow any arbitrary strategy). In addition, an adversary may be limited to polynomial-time (yielding the computational setting) or unbounded (yielding the information-theoretic setting). Finally, the adversary may be static (meaning that the set of corrupted parties is fixed before the protocol execution begins) or adaptive (meaning that the adversary can adaptively choose to corrupt throughout the protocol execution).

Wide reaching feasibility results regarding secure multi-party computation were presented in the mid to late 1980's. Let n be the number of participating parties and let t denote a bound on the number of corrupted parties. The most central and basic feasibility results are as follows:

1. For $t < n/3$, secure multi-party protocols (with fairness and guaranteed output delivery) can be achieved for any function in a point-to-point network and without any setup assumptions. This can be achieved both in the computational setting [18] (assuming the existence of enhanced trapdoor permutations), and in the information-theoretic setting with ideal private channels [4, 11].

2. For $t < n/2$, secure multi-party protocols (with fairness and guaranteed output delivery) can be achieved for any function assuming that the parties have access to a broadcast channel. This can be achieved in the computational setting [18] (with the same assumptions as above), and in the information-theoretic setting with statistical security with ideal private channels [26, 2].

3. For $t \geq n/2$, secure multi-party protocols (without fairness or guaranteed output delivery) can be achieved assuming the existence of enhanced trapdoor permutations [28, 18, 17].

1.2 The BGW Protocol

One of the most fundamental and celebrated results of the field of secure computation is the BGW protocol [4]. BGW showed that every functionality can be computed with *perfect security* in the presence of semi-honest adversaries corrupting a minority of parties, and in the presence of malicious adversaries corrupting up to a third of the parties. The discovery that secure computation can be carried out information theoretically, and the techniques used by BGW, were greatly influential.

In addition, as we shall see, the fact that security is *perfect* – informally meaning that there is a *zero probability* of cheating by the adversary – provides real security advantages over protocols that have a negligible probability of failure.

The BGW protocol builds on the GMW protocol [18] that considers secure multiparty computation in the computational setting. On a high level, the protocol works by having the parties compute the function f (from n inputs to n outputs) via an arithmetic circuit computing f. In this computation, the parties compute shares of the output of a circuit gate given shares of the input wires of that gate. To be more exact, the parties first share their inputs to each other using Shamir's secret sharing [27]; in the case of malicious adversaries, a *verifiable* secret sharing protocol is used. The parties then compute each gate of the circuit, computing Shamir shares of the output of each gate from the Shamir shares of the inputs. As we shall see, this secret sharing has the property that addition gates in the circuit can be computed using local computation only. Thus, the parties only interact in order to compute multiplication gates; this step is the most involved part of the protocol. Finally, the parties reconstruct the secrets from the shares on the output wires of the circuit in order to obtain output.

We proceed to describe the protocol in a bit more detail. Shamir's secret sharing enables the sharing of a secret s amongst n parties, so that any subset of $t+1$ or more parties can reconstruct the secret, and any subset of t or less parties learn no information whatsoever about the secret. Let $\mathbb{F}$ be a finite field of size at least n, let $\alpha_1, \ldots, \alpha_n$ be n distinct field elements, and let $s \in \mathbb{F}$. Then, in order to share s, a random polynomial $p(x) \in \mathbb{F}[x]$ of degree t and with free coefficient s is randomly chosen, and the share of the ith party P_i is set to $p(\alpha_i)$. By interpolation, given any $t+1$ points it is possible to reconstruct p and compute $s = p(0)$. Furthermore, since p is random, any t or less points give no information about s.

Now, let n denote the number of parties participating in the multiparty computation, and let t be a bound on the number of corrupted parties. The first step of the BGW protocol is for all parties to share their inputs using the Shamir secret sharing scheme. In the case of semi-honest adversaries, plain Shamir sharing with a threshold $t < n/2$ is used, and in the case of malicious adversaries verifiable secret sharing (VSS) with a threshold $t < n/3$ is used. A verifiable secret sharing protocol is needed for the case of malicious adversaries in order to prevent cheating (e.g., a party may define the sharing via a polynomial of degree higher than t). The BGW paper was also the first to construct a *perfect* VSS protocol.

Next, the parties compute the gates of the circuit. The first observation is that addition gates can be computed locally. That is, given shares $p(\alpha_i), q(\alpha_i)$ of two input wires to an addition gate, it holds that $r(\alpha_i) = p(\alpha_i) + q(\alpha_i)$ is a valid sharing of the output wire. This is due to the fact that the polynomial $r(x)$ defined by the sum of the shares has the same degree as both $p(x)$ and $q(x)$, and $r(0) = p(0) + q(0)$, implying that the free coefficient of $r(x)$ is the sum of the free coefficients of $p(x)$ and $q(x)$. We conclude that shares of the output of an addition gate can be computed by the parties without any interaction.

Regarding multiplication gates, observe that by computing $r(\alpha_i) = p(\alpha_i) \cdot q(\alpha_i)$ the parties obtain shares of a polynomial $r(x)$ with free coefficient $p(0) \cdot q(0)$ as desired. However, the degree of $r(x)$ is $2t$, since the degrees of $p(x)$ and $q(x)$ are both t. This causes a problem because additional multiplications will result in a polynomial of degree n or greater, which cannot be reconstructed (and in fact is not even fully defined) with only n shares. Thus, the multiplication protocol works by *reducing the degree* of the polynomial $r(x)$ back to t. In the case of semi-honest parties, the degree reduction can be carried out as long as $t < n/2$ (it is required that $t < n/2$ since otherwise the degree of $r(x)$ will be greater than or equal to n, which as we have mentioned is not fully defined by the n parties' shares). In the case of malicious parties, the degree reduction is much more complex and works as long as $t < n/3$. In order to obtain some intuition as to why it is needed than $t < n/3$, observe that a Shamir secret sharing can also be viewed as a Reed-Solomon code of the polynomial [23]. With a polynomial of degree t, it is possible to correct up $(n-t-1)/2$ errors. Setting $t < n/3$, we have that $n \geq 3t+1$ and so $(n-t-1)/2 \geq t$ errors can be corrected. This means that if the up to t malicious parties send incorrect values, the honest parties can use error correction and recover. Indeed, the BGW protocol in the case of malicious adversaries relies heavily on the use of error correction in order to prevent the adversary from cheating.

We remark that $t < n/3$ is not merely a limitation of the way the BGW protocol works. In particular, the fact that at most $t < n/3$ corruptions can be tolerated in the malicious model follows immediately from the fact that at most $t < n/3$ corruptions can be tolerated for Byzantine agreement [25]. In contrast, given an actual broadcast channel, it *is* possible to securely compute any functionality with information-theoretic (statistical) security [26, 2].

1.3 Our Results

Despite the importance of the BGW result, a full proof of its correctness has never appeared. In addition, a full detailed specification of the protocol in the malicious setting was also never published; this is especially apparent in the multiplication protocol for the case of any $t < n/3$ where a new step is needed. In this paper we remedy this situation and provide a full specification and proof of the BGW protocols, for both the semi-honest and malicious settings. We prove security relative to the ideal/real definition of security for multiparty computation. Among other things, this also involves carefully defining the functionalities and sub-functionalities that are used in order to achieve the result.

Our main result is a proof of the following informally stated theorem:

Theorem 1 (basic security of the BGW protocol – informally stated): *Consider a synchronous network with pairwise private channels. Then:*

1. Semi-honest: *For every n-ary functionality f, there exists a protocol for computing f with perfect security in the presence of a static semi-honest adversary corrupting up to $t < n/2$ parties;*

2. Malicious: *For every n-ary functionality f, there exists a protocol for computing f with perfect security in the presence of a static malicious adversary corrupting up to $t < n/3$.*

The communication complexity of the protocol is $O(\mathsf{poly}(n) \cdot |C|)$ where C is an arithmetic circuit computing f, and the round complexity is $O(d)$ where d is the depth of the circuit C.

All of our protocols for the case of malicious adversaries are presented in a model with pairwise private channels *and* secure broadcast. Since we only consider the case of $t < n/3$ malicious corruptions, secure broadcast can be achieved in a synchronous network with pairwise channels by running Byzantine Generals [25, 21, 15]. In order to obtain round complexity $O(d)$, the Byzantine Generals protocol of [15] must be used.

Adaptive security and composition. Theorem 1 is proven in the classic setting of static corruptions and stand-alone computation, where the latter means that security is proven for the case that only a single protocol execution takes place. Fortunately, using theorems proven in [7] and [20] regarding properties of protocols that achieve perfect security, we are able to formally derive security in the presence of adaptive adversaries and under universal composability [6] (where security is guaranteed to hold when many arbitrary protocols are run concurrently with the secure protocol). We therefore have the following corollary, that relates to a far more powerful adversarial setting:

Corollary 2 (UC information-theoretic security of the BGW protocol): *Consider a synchronous network with private channels. Then, for every n-ary functionality f, there exists a protocol for computing f with perfect universally composable security in the presence of an adaptive semi-honest adversary corrupting up to $t < n/2$ parties, and there exists a protocol for computing f with perfect universally composable security in the presence of an adaptive malicious adversary corrupting up to $t < n/3$ parties.*

Corollary 2 refers to information-theoretic security in the ideal private channels model. We now derive a corollary to the computational model with authenticated channels only. In order to derive this corollary, we first observe that information-theoretic security implies security in the presence of polynomial-time adversaries (this holds as long as the simulator is required to run in time that is polynomial in the running time of the adversary, as advocated in [17, Sec. 7.6.1]). Furthermore, the ideal private channels of the information-theoretic setting can be replaced with computationally secure channels that can be constructed over authenticated channels using semantically secure public-key encryption (for the case of static corruptions) and non-committing public-key encryption [8] (for the case of adaptive corruptions). We have:

Corollary 3 (UC computational security of the BGW protocol): *Consider a synchronous network with authenticated channels. Assuming the existence of semantically secure public-key encryption (resp., non-committing encryption), for every n-ary functionality f there exists a protocol for computing f with universally composable security in the presence of a static (resp., adaptive) malicious adversary corrupting up to $t < n/3$ parties.*

We stress that unlike the UC-secure computational protocols of [10], the protocols of Corollary 3 are in the *plain model*, with authenticated channels and no other trusted setup (in particular, no common reference string). Although well accepted folklore, none of the above has actually ever been proven in full. Thus, our work also constitutes the first full proof that universally composable protocols exist in the plain model (with authenticated channels) for any functionality, in the presence of (adaptive or static) malicious adversaries corrupting up to any $t < n/3$ parties.

Organization. In Section 2, we present a brief overview of the definitions of perfectly secure multiparty computation and of the modular sequential composition theorem that is used throughout in our proofs. Then, in Section 3, we describe Shamir's secret sharing scheme and rigorously prove a number of useful properties of this scheme. In Section 4 we present the BGW protocol for the case of semi-honest adversaries. An overview of the overall construction appears in Section 4.1, and an overview of the multiplication protocol appears at the beginning of Section 4.3.

The BGW protocol for the case of malicious adversaries is presented in Sections 5 to 7. In Section 5 we present the BGW verifiable secret sharing (VSS) protocol that uses bivariate polynomials. This section includes background on Reed-Solomon encoding and properties of bivariate polynomials that are needed for proving the security of the VSS protocol. Next, in Section 6 we present the most involved part of the protocol – the multiplication protocol for computing shares of the product of shares. This involves a number of steps and subprotocols, some of which are new; most notably, a protocol for securely evaluating a shared polynomial on a predetermined point which is used for processing complaints in the multiplication step. The main tool of the BGW multiplication protocol is a subprotocol for verifiably sharing the product of a party's shares. This subprotocol, along with a detailed discussion and overview, is presented in Section 6.3. Our aim has been to prove the security of the original BGW protocol. However, where necessary, some changes were made to the multiplication protocol as described originally in [4]. Finally, in Section 7, the final protocol for secure multiparty computation is presented. The protocol is proven secure for any VSS and multiplication protocols that securely realize the VSS and multiplication functionalities that we define in Sections 5 and 6, respectively.

We conclude by showing how to derive security in other settings (adaptive adversaries, composition, and the computational setting) in Section 8, and with an exact count of the communication complexity of the BGW protocol for malicious adversaries in Appendix A.

2 Preliminaries and Definitions

In this section, we briefly review the definition of perfect security in the presence of semi-honest and malicious adversaries. We refer the reader to [17, Sec. 7.6.1] and [5] for detailed definitions and explanations.

In the definitions below, we consider the *stand-alone* setting with a *synchronous* network, and perfectly *private channels* between all parties. For simplicity, we will also assume that the parties have a broadcast channel; as is standard, this can be implemented using an appropriate Byzantine Generals protocol [25, 21]. Since we consider synchronous channels and the computation takes place in clearly defined rounds, if a message is not received in a given round then this fact is immediately known to the party who is supposed to receive the message. Thus, we can write "if a message is not received" or "if the adversary does not send a message" and this is well defined. We consider *static corruptions* meaning that the set of corrupted parties is fixed ahead of time, and the *stand-alone setting* meaning that only a single protocol execution takes place; extensions to the case of adaptive corruptions and composition are considered in Section 8.

Basic notation. We denote the number of parties by n, and a bound on the number of corrupted parties by t. Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be a possibly probabilistic n-ary functionality, where $f_i(x_1, \ldots, x_n)$ denotes the ith element of $f(x_1, \ldots, x_n)$. We denote by $I = \{i_1, \ldots i_t\} \subset [n]$ the indices of the corrupted parties. By the above, $|I| \leq t$.

2.1 Perfect Security in the Presence of Semi-Honest Adversaries

Let $\vec{x} = (x_1, \ldots, x_n)$, and let $\vec{x}_I$ and $f_I(\vec{x})$ denote projections of the corresponding n-ary sequence on the coordinates in I; that is, $\vec{x}_I = (x_{i_1}, \ldots, x_{i_t})$ and $f_I(\vec{x}) = (f(\vec{x})_{i_1}, \ldots, f(\vec{x})_{i_t})$ where $I = \{i_1, \ldots, i_t\}$. The view of the ith party P_i during an execution of a protocol π on inputs $\vec{x}$, denoted

$\text{VIEW}_i^\pi(\vec{x})$, is defined to be $(x_i, r_i; m_1, \ldots, m_\ell)$ where x_i is P_i's private input, r_i is its internal coin tosses, and m_j is the jth message that it received in the protocol execution. For every $I = \{i_1, \ldots i_t\}$, we denote $\text{VIEW}_I^\pi(\vec{x}) = (\text{VIEW}_{i_1}^\pi(\vec{x}), \ldots \text{VIEW}_{i_t}^\pi(\vec{x}))$. The output of all parties from an execution of π on inputs $\vec{x}$ is denoted $\text{OUTPUT}^\pi(\vec{x})$; observe that the output of each party can be computed from its own (private) view of the execution.

We are now ready to define security in the presence of semi-honest adversaries; this is called t-privacy in order to differentiate it from t-security that is used for the case of malicious adversaries. Since we only deal with perfect security in this paper, we use the terms t-private and t-secure without any additional adjective, with the understanding that the privacy/security is perfect.

Definition 2.1 (t-privacy of n-party protocols – general case): *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be a probabilistic n-ary functionality and let π be a protocol. We say that π t-privately computes f if there exists a probabilistic polynomial-time algorithm $\mathcal{S}$ such that for every $I \subset [n]$ of cardinality at most t, and every $\vec{x} \in (\{0,1\}^*)^n$ where $|x_1| = \ldots = |x_n|$, it holds that:*

$$\left\{(S(I, \vec{x}_I, f_I(\vec{x})), f(\vec{x}))\right\} \equiv \left\{(\text{VIEW}_I^\pi(\vec{x}), \text{OUTPUT}^\pi(\vec{x}))\right\}. \tag{1}$$

Observe that Definition 2.1 requires that the *joint* distribution of the output of $\mathcal{S}$ and all outputs of the functionality is identically distributed to the joint distribution of the view of the corrupted parties and all of the parties' outputs. This is necessary for probabilistic functionalities, as explained in [5, 17]. However, for the case of deterministic functionalities it suffices to separately require correctness (meaning that the parties always output the correct result) and that the output of $\mathcal{S}$ is identically distributed to the view of the corrupted parties in the protocol.

Definition 2.2 (t-privacy of n-party protocols – deterministic case): *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be a* deterministic *n-ary functionality and let π be a protocol. We say that π t-privately computes f if for every $\vec{x} \in (\{0,1\}^*)^n$ where $|x_1| = \ldots = |x_n|$,*

$$\text{OUTPUT}^\pi(x_1, \ldots, x_n) = f(x_1, \ldots, x_n)$$

and there exists a probabilistic polynomial-time algorithm $\mathcal{S}$ such that for every $I \subset [n]$ of cardinality at most t, and every $\vec{x} \in (\{0,1\}^*)^n$ where $|x_1| = \ldots = |x_n|$, it holds that:

$$\left\{S\left(I, \vec{x}_I, f_I\left(\vec{x}\right)\right)\right\} \equiv \left\{\text{VIEW}_I^\pi(\vec{x})\right\} \tag{2}$$

The BGW protocol can be used to t-privately compute any deterministic or probabilistic functionality that can be computed in probabilistic polynomial-time.[1] Nevertheless, we prove security only for deterministic functionalities. This simplifies our proof because we are then able to use Definition 2.2 in order to prove security. Furthermore, this suffices because it is possible to t-privately compute any probabilistic functionality using a general protocol for t-privately computing any deterministic functionality; see [17, Sec. 7.3.1] for a proof.

[1]Indeed, it is even possible to t-privately compute functionalities that are not efficiently computable. However, in such a case, the simulator $\mathcal{S}$ cannot be probabilistic polynomial-time.

2.2 Perfect Security in the Presence of Malicious Adversaries

We now consider malicious adversaries that can follow any arbitrary strategy in order to carry out their attack; we stress that the adversary is not required to be efficient in any way. Security is formalized by comparing a real protocol execution to an ideal model where the parties just send their inputs to the trusted party and receive back outputs. See [5, 17] for details on how to define these real and ideal executions.

Let f be as above and let π be a n-party protocol computing f. Let $\mathcal{A}$ be an arbitrary machine with auxiliary input z, and denote by $\mathcal{S}$ an ideal-model adversary/simulator. We denote by $\text{REAL}_{\pi,\mathcal{A}(z),I}(\vec{x})$ the random variable consisting of the view of the adversary $\mathcal{A}$ controlling the corrupted parties in I and the outputs of the honest parties, following a real execution of π where for every $i \in [n]$, party P_i has input x_i. We stress that the execution takes place in a synchronous network with private point-to-point channels. We denote by $\text{IDEAL}_{f,\mathcal{S}(z),I}(\vec{x})$ the analogous outputs of the ideal adversary $\mathcal{S}$ and honest parties after an ideal execution with a trusted party computing f. Finally, we require that $\mathcal{S}$ run in time that is polynomial in the running time of $\mathcal{A}$, whatever the latter may be. As argued in [5, 17] this is important since it guarantees that information-theoretic security implies computational security. In such a case, we say that $\mathcal{S}$ is of comparable complexity to $\mathcal{A}$.

Definition 2.3 *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-ary functionality and let π be a protocol. We say that π t-securely computes f if for every probabilistic adversary $\mathcal{A}$ in the real model, there exists a probabilistic adversary $\mathcal{S}$ of comparable complexity in the ideal model, such that for every $I \subset [n]$ of cardinality at most t, every $\vec{x} \in (\{0,1\}^*)^n$ where $|x_1| = \ldots = |x_n|$, and every $z \in \{0,1\}^*$, it holds that:*

$$\left\{\text{IDEAL}_{f,\mathcal{S}(z),I}(\vec{x})\right\} \equiv \left\{\text{REAL}_{\pi,\mathcal{A}(z),I}(\vec{x})\right\}.$$

Reactive functionalities. The above definition refers to functionalities that map inputs to outputs in a single computation. However, some computations take place in stages, and state is preserved between stages. Two examples of such functionalities are mental poker (where cards are dealt and thrown and redealt) and commitment schemes (where there is a separate commitment and decommitment phase). Such functionalities are called reactive, and the definition of security is extended to this case in the straightforward way by allowing the trusted party to obtain inputs and send outputs in phases. As we will see, some of the subprotocols that are used in the BGW protocol for the case of malicious adversaries seem to be only possible to prove when defined in a reactive way. In such cases, we refer to the reactive analogue of the above definition.

2.3 Modular Composition

The sequential modular composition theorem [5] states that in order to analyze the security of a protocol π_f for computing f that uses a subprotocol π_g for computing g, it suffices to consider the execution of π_f in a model where a trusted third party is used to ideally compute g (instead of the parties running the real subprotocol π_g). This theorem facilitates a modular analysis of security: first prove the security of π_g and then prove the security of π_f using an ideal party for g. The model in which π_f is analyzed using ideal calls to g, instead of executing π_g, is called the *g-hybrid model* because it involves both a real protocol execution and an ideal trusted third party computing g. More formally, in the hybrid model, the parties all have oracle-tapes for some oracle (trusted party) that computes the functionality g. Then, if the real protocol π_f instructs the parties to run the subprotocol π_g using inputs $\alpha_1, \ldots, \alpha_n$, then each party P_i simply writes α_i to its outgoing oracle tape. Then, in the next round, it receives back the output $g_i(\alpha_1, \ldots, \alpha_n)$ on its incoming oracle tape. We denote by $\text{HYBRID}^g_{\pi_f,\mathcal{A}(z),I}(\vec{x})$ an execution of protocol π_f where each call to π_g is carried out using an oracle computing g. See [5, 17] for a formal definition of this model for both the semi-honest and malicious cases, and for proofs that if π_f t-privately (resp., t-securely) computes f in the g-hybrid model, and π_g t-privately (resp., t-securely) computes g, then π_f when run in the real model using π_g t-privately (resp., t-securely) computes f.

3 Shamir's Secret Sharing Scheme [27] and Properties

3.1 The Basic Scheme

A central tool in the BGW protocol is Shamir's secret-sharing scheme [27]. Roughly speaking, a t-out-of-n secret sharing scheme takes as input a secret s from some domain, and outputs n shares, with the property that it is possible to efficiently reconstruct s from every subset of t shares, but every subset of less than t shares reveals nothing about the secret s. The value t is called the threshold of the scheme.

A secret sharing scheme consist of two algorithm: the first algorithm, called the sharing algorithm, takes as input the secret s and the parameters t and n, and outputs n shares. The second algorithm, called the reconstruction algorithm, takes as input t shares and outputs a value s. It is required that the reconstruction of shares generated from a value s yields the same value s.

Informally, Shamir's secret-sharing scheme works as follows. Let $\mathbb{F}$ be a finite field of size greater than n and let $s \in \mathbb{F}$. The sharing algorithm defines a polynomial $q(x)$ of degree $t-1$ in $\mathbb{F}[x]$, such that its free coefficient is the secret s and all the other coefficients are selected uniformly and independently at random in $\mathbb{F}$.[2] Finally, the shares are defined to be $q(\alpha_i)$ for every $i \in \{1, \ldots, n\}$, where $\alpha_1, \ldots, \alpha_n$ are any n distinct predetermined values in $\mathbb{F}$. The reconstruction algorithm of this scheme is based on the fact that any t points define exactly one polynomial of degree $t-1$. Therefore, using interpolation it is possible to efficiently reconstruct the polynomial $q(x)$ given any subset of t points $(\alpha_i, q(\alpha_i))$ as output by the sharing algorithm. Then, given $q(x)$ it is possible to simply compute $s = q(0)$.

In order to see that any subset of less than t shares reveals nothing about s, observe that for every set of $t-1$ points $(\alpha_i, q(\alpha_i))$ and every possible secret $s' \in \mathbb{F}$, there exists a unique polynomial $q'(x)$ such that $q'(0) = s'$ and $q'(\alpha_i) = q(\alpha_i)$. Since the polynomial is chosen randomly by the sharing algorithm, there is the same likelihood that the underlying polynomial is $q(x)$ (and so the secret is s) and that the polynomial is $q'(x)$ (and so the secret is s'). We now formally describe the scheme.

Shamir's t-out-of-n secret sharing scheme. Let $\mathbb{F}$ be a finite field of order p where $p > n$, and let $\alpha_1, \ldots, \alpha_n$ be any *distinct non-zero* elements of $\mathbb{F}$.

- **The sharing algorithm:** Let $\mathsf{share}(s, t, n, \vec{\alpha})$ be the algorithm that receives for input s, t, n and $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n)$ where $s, \alpha_1, \ldots, \alpha_n \in \mathbb{F}$ and $t \leq n$. Then, share chooses $t-1$ random

[2]Throughout, when we refer to a polynomial of degree t, we mean of degree *at most* t.

values $q[1],\dots q[t-1] \leftarrow \mathbb{F}$, independently and uniformly distributed in $\mathbb{F}$, and defines the polynomial:

$$q(x) = s + q[1]x + \dots q[t-1]x^{t-1}$$

where all calculations are in the field $\mathbb{F}$ (e.g., if $\mathbb{F}$ is the field $\mathbb{Z}_p$ for some prime p, then all the operations are modulo p). Finally, share computes the output shares $q(\alpha_1),\dots,q(\alpha_n)$, where $q(\alpha_i)$ is the share of party P_i.

- **The reconstruction algorithm:** Algorithm reconstruct$((\alpha_{i_1},\beta_{i_1}),\dots,(\alpha_{i_t},\beta_{i_t}))$ receives t points and finds the unique polynomial $q(x)$ of degree $t-1$ such that for every $j=1,\dots,t$ it holds that $q(\alpha_{i_j}) = \beta_{i_j}$. The algorithm then outputs the coefficients of the polynomial $q(x)$ (note that the original secret can be obtained by simply computing $s = q(0)$).

Notation. Let $\mathcal{P}^{s,t}$ be the set of all polynomials with degree less than or equal to t with free coefficient s. We stress that polynomials from $\mathcal{P}^{s,t}$ are associated with a secret sharing scheme with threshold $\boldsymbol{t+1}$ because their degree is t and not $t-1$. We will use the threshold $t+1$ here, because this is the threshold that we need later for the secure computation protocol with t corrupt parties. Observe that for every two values $s,s' \in \mathbb{F}$, it holds that $|\mathcal{P}^{s,t}| = |\mathcal{P}^{s',t}| = |\mathbb{F}|^t$.

3.2 Basic Properties

In this section, we formally prove some basic properties of Shamir's secret sharing scheme. We first show that a single point of a polynomial chosen at random from $\mathcal{P}^{s,t}$ is distributed uniformly at random in $\mathbb{F}$; this can be generalized to hold for any t points.

Claim 3.1 *For every $t \geq 1$, and for every $s,\alpha,y \in \mathbb{F}$ with $\alpha \neq 0$, it holds that:*

$$\Pr_{q \in_R \mathcal{P}^{s,t}}[q(\alpha) = y] = \frac{1}{|\mathbb{F}|}.$$

Proof: Fix s, y and α with $\alpha \neq 0$. Denote the ith coefficient of the polynomial $q(x)$ by $q[i]$, for $i=1,\dots,t$. Then:

$$\Pr[q(\alpha) = y] \quad = \quad \Pr\left[y = s + \sum_{i=1}^{t} q[i]\alpha^i\right] = \Pr\left[y = s + q[1]\alpha + \sum_{i=2}^{t} q[i]\alpha^i\right]$$

where the probability is taken over the random choice of $q \in_R \mathcal{P}^{s,t}$, or equivalently of the coefficients $q[1],\dots,q[t] \in_R \mathbb{F}$. Fix $q[2],\dots,q[t]$ and denote $v = \sum_{i=2}^{t} q[i]\alpha^i$. Then, for a randomly chosen $q[1] \in_R \mathbb{F}$ we have that

$$\begin{aligned}
\Pr\Big[q(\alpha) = y\Big] &= \Pr\Big[y = s + q[1]\alpha + v\Big] \\
&= \Pr\Big[q[1]\alpha = y - s - v\Big] \\
&= \Pr\Big[q[1] = \alpha^{-1}\cdot(y - s - v)\Big] \\
&= \frac{1}{|\mathbb{F}|}
\end{aligned}$$

where the third equality holds since $\alpha \in \mathbb{F}$ and $\alpha \neq 0$ implying that α has an inverse, and the last equality is due to the fact that $q[1] \in_R \mathbb{F}$ is randomly chosen. ■

In the protocol for secure computation, a dealer hides a secret s by choosing a polynomial $f(x)$ at random from $\mathcal{P}^{s,t}$, and each party P_i receives a share, which is a point $f(\alpha_i)$. In this context, the adversary controls a subset of at most t parties, and thus receives at most t shares. We now show that any subset of at most t shares do not reveal *any* information about the secret. In Section 3.1, we explained intuitively why the above holds. We now formally prove this claim, by stating that the distribution $\{f(\alpha_i)\}_{i\in I}$ is identical to the distribution $\{g(\alpha_i)\}_{i\in I}$, for any subset $I \subset [n]$, where $|I| \leq t$ and $f(x) \in_R \mathcal{P}^{s,t}$, $g(x) \in_R \mathcal{P}^{s',t}$ for any fixed $s, s' \in \mathbb{F}$.

Claim 3.2 *For any set of distinct non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$, any pair of values $s, s' \in \mathbb{F}$, any subset $I \subset [n]$ where $|I| = \ell \leq t$, and every $\vec{y} \in \mathbb{F}^\ell$ it holds that:*

$$\Pr_{f(x)\in_R\mathcal{P}^{s,t}}\left[\vec{y} = \left(\{f(\alpha_i)\}_{i\in I}\right)\right] = \Pr_{g(x)\in_R\mathcal{P}^{s',t}}\left[\vec{y} = \left(\{g(\alpha_i)\}_{i\in I}\right)\right] = \frac{1}{|\mathbb{F}|^\ell}$$

where $f(x)$ and $g(x)$ are chosen uniformly and independently at random from $\mathcal{P}^{s,t}$ and $\mathcal{P}^{s',t}$, respectively.

Proof: We first prove the claim for the special case that $\ell = t$. Fix $s, s' \in \mathbb{F}$, fix non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$, and fix $I \subset [n]$ with $|I| = t$. Moreover, fix $\vec{y} \in \mathbb{F}^t$. Let y_i be the ith element of the vector $\vec{y}$ for every $i \in \{1, \ldots, t\}$. We now show that:

$$\Pr_{f(x)\in_R\mathcal{P}^{s,t}}\left[\vec{y} = \left(\{f(\alpha_i)\}_{i\in I}\right)\right] = \frac{1}{|\mathcal{P}^{s,t}|}.$$

The values of $\vec{y}$ define a unique polynomial from $\mathcal{P}^{s,t}$. This is because there exists a single polynomial of degree t that passes through the points $(0, s)$ and $\{(\alpha_i, y_i)\}_{i\in I}$. Let $f'(x)$ be this unique polynomial. By definition we have that $f'(x) \in \mathcal{P}^{s,t}$ and so:

$$\Pr_{f(x)\in_R\mathcal{P}^{s,t}}\left[\vec{y} = \left(\{f(\alpha_i)\}_{i\in I}\right)\right] = \Pr\left[f(x) = f'(x)\right] = \frac{1}{|\mathcal{P}^{s,t}|}$$

where the latter is true since $f(x)$ is chosen uniformly at random from $\mathcal{P}^{s,t}$, and $f'(x)$ is a fixed polynomial in $\mathcal{P}^{s,t}$.

Using the same reasoning, and letting $g'(x)$ be the unique polynomial that passes through the points $(0, s')$ and $\{(\alpha_i, y_i)\}_{i\in I}$ we have that:

$$\Pr_{g(x)\in_R\mathcal{P}^{s',t}}\left[\vec{y} = \left(\{g(\alpha_i)\}_{i\in I}\right)\right] = \Pr\left[g(x) = g'(x)\right] = \frac{1}{|\mathcal{P}^{s',t}|}.$$

The proof for the case of $\ell = t$ is concluded by observing that for every s and s' in $\mathbb{F}$, it holds that $|\mathcal{P}^{s,t}| = |\mathcal{P}^{s',t}| = |\mathbb{F}|^t$, and so:

$$\Pr_{f(x)\in_R\mathcal{P}^{s,t}}\left[\vec{y} = \left(\{f(\alpha_i)\}_{i\in I}\right)\right] = \Pr_{g(x)\in_R\mathcal{P}^{s',t}}\left[\vec{y} = \left(\{g(\alpha_i)\}_{i\in I}\right)\right] = \frac{1}{|\mathbb{F}|^t}.$$

For the general case where $|I| = \ell$ may be less than t, fix $J \subset [n]$ with $|J| = t$ and $I \subset J$. Observe that for every vector $\vec{y} \in \mathbb{F}^\ell$:

$$\Pr_{f(x)\in_R\mathcal{P}^{s,t}}\left[\vec{y} = \left(\{f(\alpha_i)\}_{i\in I}\right)\right] = \sum_{\vec{y}'\in\mathbb{F}^{t-\ell}} \Pr\left[(\vec{y}, \vec{y}') = \left(\{f(\alpha_i)\}_{i\in I}, \{f(\alpha_j)\}_{j\in J\setminus I}\right)\right] = |\mathbb{F}|^{t-\ell} \cdot \frac{1}{|\mathbb{F}|^t} = \frac{1}{|\mathbb{F}|^\ell}.$$

This holds for both s and s' and so the proof is concluded. ■

As a corollary, we have that any $\ell \leq t$ points on a random polynomial are uniformly distributed in the field $\mathbb{F}$. This follows immediately from Claim 3.2 because stating that every $\vec{y}$ appears with probability $1/|\mathbb{F}|^{\ell}$ is equivalent to stating that the shares are uniformly distributed. That is:

Corollary 3.3 *For any secret $s \in \mathbb{F}$, any set of distinct non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$, and any subset $I \subset [n]$ where $|I| = \ell \leq t$, it holds that:*

$$\left\{ \{f(\alpha_i)\}_{i \in I} \right\} \equiv \left\{ U_{\mathbb{F}}^{(1)}, \ldots, U_{\mathbb{F}}^{(\ell)} \right\}$$

where $f(x)$ is chosen uniformly at random from $\mathcal{P}^{s,t}$ and $U_{\mathbb{F}}^{(1)}, \ldots, U_{\mathbb{F}}^{(\ell)}$ are ℓ independent random variables that are uniformly distributed over $\mathbb{F}$.

3.3 Multiple Polynomials

In the protocol for secure computation, parties hide secrets and distribute them using Shamir's secret sharing scheme. As a result, the adversary receives $m \cdot |I|$ shares, $\{f_1(\alpha_i), \ldots, f_m(\alpha_i)\}_{i \in I}$, for some value m. The secrets $a_1, \ldots, a_m$ may not be independent. We therefore need to show that the shares that the adversary receives for all secrets do not reveal any information about any of the secrets. Intuitively, this follows from the fact that Claim 3.2 is stated for *any* two secrets s, s', and in particular for two secrets that are known and may be related. Formally:

Claim 3.4 *For any $m \in \mathbb{N}$, any set of non-zero distinct values $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$, any two sets of secrets $(a_1, \ldots, a_m) \in \mathbb{F}^m$ and $(b_1, \ldots, b_m) \in \mathbb{F}^m$, and any subset $I \subset [n]$ of size $|I| \leq t$, it holds that:*

$$\left\{ \{(f_1(\alpha_i), \ldots, f_m(\alpha_i))\}_{i \in I} \right\} \equiv \left\{ \{(g_1(\alpha_i), \ldots, g_m(\alpha_i))\}_{i \in I} \right\}$$

where for every j, $f_j(x)$, $g_j(x)$ are chosen uniformly at random from $\mathcal{P}^{a_j,t}$ and $\mathcal{P}^{b_j,t}$, respectively.

Proof Sketch: Define the following hybrid ensemble, for $k = 0, \ldots, m$:

$$H^{(k)} = \left\{ \{(f_1(\alpha_i), \ldots, f_k(\alpha_i), g_{k+1}(\alpha_i), \ldots, g_m(\alpha_i))\}_{i \in I} \right\}$$

First, observe that $H^{(1)} = \{\{(g_1(\alpha_i), \ldots, g_m(\alpha_i))\}_{i \in I}\}$ and $H^{(m)} = \{\{(f_1(\alpha_i), \ldots, f_m(\alpha_i))\}_{i \in I}\}$. Next, we claim that $H^{(k)} \equiv H^{(k+1)}$ for every $k = 0, \ldots, m-1$. This follows immediately from Claim 3.2 because the only difference between $H^{(k)}$ and $H^{(k+1)}$ is the $(k+1)$th element: in $H^{(k)}$ it is $\{g_{k+1}(\alpha_i)\}_{i \in I}$, whereas in $H^{(k+1)}$ it is $\{f_{k+1}(\alpha_i)\}_{i \in I}$), and by Claim 3.2 we know that $\{g_{k+1}(\alpha_i)\}_{i \in I} \equiv \{f_{k+1}(\alpha_i)\}_{i \in I}$. We therefore conclude that $H^{(0)} \equiv H^{(m)}$, and the claim follows. ■

3.4 Hiding the Leading Coefficient

In Shamir's secret sharing scheme, the dealer creates shares by constructing a polynomial of degree t, where its free coefficient is fixed and all the other coefficients are chosen uniformly at random. In Claim 3.2 we showed that any t or fewer points on such a polynomial do not reveal any information about the fixed coefficient which is the free coefficient.

We now consider this claim when we choose the polynomial differently. In particular, we now fix the *leading* coefficient of the polynomial (i.e., the coefficient of the monomial x^t), and choose all the other coefficients uniformly and independently at random, including the free coefficient. As in the previous section, we show that any subset of t or fewer points on such a polynomial do not reveal any information about the fixed coefficient, which in this case is the leading coefficient. We will need this claim for proving the security of one of the sub-protocols for the malicious case (in Section 6.3).

Let $\mathcal{P}^{\mathsf{lead}}_{s,t}$ be the set of all the polynomials of degree t with *leading* coefficient s. Namely, the polynomials have the structure: $f(x) = a[0] + a[1]x + \ldots a[t-1]x^{t-1} + sx^t$. The following claim is analogous to Claim 3.2:

Claim 3.5 *For any set of distinct non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$, any pair of values $s, s' \in \mathbb{F}$, any subset $I \subset [n]$ where $|I| = \ell \leq t$, and every $\vec{y} \in \mathbb{F}^\ell$ it holds that:*

$$\Pr_{f(x) \in_R \mathcal{P}^{\mathsf{lead}}_{s,t}}\left[\vec{y} = \left(\{f(\alpha_i)\}_{i \in I}\right)\right] = \Pr_{g(x) \in_R \mathcal{P}^{\mathsf{lead}}_{s',t}}\left[\vec{y} = \left(\{g(\alpha_i)\}_{i \in I}\right)\right] = \frac{1}{|\mathbb{F}|^\ell}$$

The proof of Claim 3.5 is almost identical to that of Claim 3.2 and is therefore omitted. The following corollary is derived from Claim 3.5 in the same way that Corollary 3.3 is derived from Claim 3.2.

Corollary 3.6 *For any secret $s \in \mathbb{F}$, any set of distinct non-zero elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$, and any subset $I \subset [n]$ where $|I| = \ell \leq t$, it holds that:*

$$\left\{ \{f(\alpha_i)\}_{i \in I} \right\} \equiv \left\{ U_{\mathbb{F}}^{(1)}, \ldots, U_{\mathbb{F}}^{(\ell)} \right\}$$

where $f(x)$ is chosen uniformly at random from $\mathcal{P}^{\mathsf{lead}}_{s,t}$ and $U_{\mathbb{F}}^{(1)}, \ldots, U_{\mathbb{F}}^{(\ell)}$ are ℓ independent random variables that are uniformly distributed over $\mathbb{F}$.

3.5 Matrix Representation

In this section we present a useful representation for polynomial evaluation. We being by defining the Vandermonde matrix for the values $\alpha_1, \ldots, \alpha_n$. As we will see, the computation of a polynomial on $\alpha_1, \ldots, \alpha_n$ can be obtained by multiplying the associated Vandermonde matrix with the vector containing the polynomial coefficients.

Definition 3.7 (Vandermonde matrix for $(\alpha_1, \ldots, \alpha_n)$) *Let $\alpha_1, \ldots, \alpha_n$ be n distinct non zero elements from $\mathbb{F}$. The Vandermonde matrix $V_{\vec{\alpha}}$ for $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n)$ is the $n \times n$ matrix over $\mathbb{F}$ defined by $V_{\vec{\alpha}}[i,j] \stackrel{\text{def}}{=} (\alpha_i)^{j-1}$. That is,*

$$V_{\vec{\alpha}} \stackrel{\text{def}}{=} \begin{pmatrix} 1 & \alpha_1 & \ldots & (\alpha_1)^{n-1} \\ 1 & \alpha_2 & \ldots & (\alpha_2)^{n-1} \\ \vdots & \vdots & & \vdots \\ 1 & \alpha_n & \ldots & (\alpha_n)^{n-1} \end{pmatrix}$$

The following fact from linear algebra will be of importance to us:

Fact 3.8 *Let $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n)$, where all α_i are distinct and non-zero. Then, $V_{\vec{\alpha}}$ is invertible.*

Matrix representation of polynomial evaluations. Let $V_{\vec{\alpha}}$ be the Vandermonde matrix for $\vec{\alpha}$ and let $q = q[0] + q[1]x + \cdots + q[t]x^t$ be a polynomial where $t < n$. Define the vector $\vec{q}$ of length n as follows: $\vec{q} \stackrel{\text{def}}{=} (q[0], \ldots q[t], 0, \ldots, 0)$. Then, it holds that:

$$V_{\vec{\alpha}} \cdot \vec{q} = \begin{pmatrix} 1 & \alpha_1 & \ldots & (\alpha_1)^{n-1} \\ 1 & \alpha_2 & \ldots & (\alpha_2)^{n-1} \\ \vdots & \vdots & & \vdots \\ 1 & \alpha_n & \ldots & (\alpha_n)^{n-1} \end{pmatrix} \cdot \begin{pmatrix} q[0] \\ \vdots \\ q[t] \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} q(\alpha_1) \\ \vdots \\ \vdots \\ q(\alpha_n) \end{pmatrix}$$

which is the evaluation of the polynomial $q(x)$ on the points $\alpha_1, \ldots, \alpha_n$.

4 The Protocol for Semi-Honest Adversaries

4.1 Overview

We now provide a high-level overview of the protocol for t-privately computing any deterministic functionality in the presence of a semi-honest adversary who has corrupted up to at most $t < n/2$ parties. Let f be the functionality that the parties wish to compute, and let C be an *arithmetic circuit* that computes f. We assume that all arithmetic operations in the circuit are carried out over a finite field $\mathbb{F}$ of size greater than n, and that the input of each party is an element from $\mathbb{F}$. In addition, we assume that the arithmetic circuit C consists of three types of gates: *addition* gates, *multiplication* gates, and *multiplication-by-a-constant* gates. We require that no circuit-output wire in the circuit is used as input into any other gate. This enables a topological sort over the (computation of the) gates and output wires of the circuit, so that the output wires come strictly after all gates.

The protocol works by having the parties jointly compute the circuit from the input gates to the output gates, under the invariant that at each stage of the computation the parties hold Shamir shares of the values on the wires up to the gates that have not yet been computed. In more detail, the protocol has three phases:

- **The input stage.** In this stage, each party creates shares of its input using Shamir's secret sharing scheme using threshold $t+1$ (for a given $t < n/2$), and distributes the shares among the parties.
- **The computation stage**. In this stage, the parties jointly compute the circuit C, gate by gate. In each step, the parties compute shares of the output of a given gate, based on the shares of the inputs to that gate that they already have. The actions of the parties in this stage depends on the type of gate being computed:
  1. *Addition gate:* Given shares of the input wires to the gate, the output is computed without any interaction by each party simply adding their local shares together. Let the inputs to the gate be a and b and let the shares of the parties be defined by two degree-t polynomials $f_a(x)$ and $f_b(x)$ (meaning that each party P_i holds $f_a(\alpha_i), f_b(\alpha_i)$ where $f_a(0) = a$ and $f_b(0) = b$). Then the polynomial $f_{a+b}(x)$ defined by shares $f_{a+b}(\alpha_i) =$

$f_a(\alpha_i) + f_b(\alpha_i)$, for every i, is a degree-t polynomial with free coefficient $a + b$. Thus, each party simply locally adds its own shares $f_a(\alpha_i), f_b(\alpha_i)$ together, and the result is that the parties hold legal shares of the sum of the inputs, as required.

2. *Multiplication-by-a-constant gate:* This type of gate can also be computed without any interaction. Let the input to the gate be a and let $f_a(x)$ be the t-degree polynomial defining the shares, as above. The aim of the parties is to obtain shares of the value $c \cdot a$, where c is the constant of the gate. Then, each party P_i holding $f_a(\alpha_i)$ simply defines its output share to be $f_{c \cdot a}(\alpha_i) = c \cdot f_a(\alpha_i)$. It is clear that $f_{c \cdot a}(x)$ is a degree-t polynomial with free coefficient $c \cdot a$, as required.
3. *Multiplication gate:* As above, let the inputs be a and b, and let $f_a(x)$ and $f_b(x)$ be the polynomials defining the shares. Here, as in the case of an addition gate, the parties can just multiply their shares together and define $h(\alpha_i) = f_a(\alpha_i) \cdot f_b(\alpha_i)$. The free coefficient of this polynomial is $a \cdot b$, as required. However, $h(x)$ may be of degree $2t$ instead of t. Furthermore, $h(x)$ is not a "random polynomial" but has a specific structure. For example, $h(x)$ is typically not irreducible (since it can be expressed as the product of $f_a(x)$ and $f_b(x)$). Thus, it does not suffice to use local computation here. The parties therefore compute this gate by running an interactive protocol that t-privately computes the multiplication functionality F_{mult}, defined by

$$F_{mult}\Big((f_a(\alpha_1), f_b(\alpha_1)), \ldots, (f_a(\alpha_n), f_b(\alpha_n))\Big) = \Big(f_{ab}(\alpha_1), \ldots, f_{ab}(\alpha_n)\Big)$$

where $f_{ab}(x)$ is a random polynomial with free coefficient $a \cdot b$ (i.e., $f_{ab}(x) \in_R \mathcal{P}^{a \cdot b, t}$).

- **The output stage.** At the end of the computation stage, the parties hold shares of the output wires. In order to obtain the actual output, the parties send their shares to one another and reconstruct the values of the output wires. Specifically, if a given output wire defines output for party P_i, then all parties send their shares of that wire value to P_i.

Organization of this section. In Section 4.2, we fully specify the above protocol and prove its security in the F_{mult}-hybrid model. (Recall that in this model, the parties exchange real protocol messages, and in addition have access to a trusted party who computes F_{mult} for them.) We also derive a corollary for t-privately computing any linear function in the plain model (i.e., without any use of the F_{mult} subfunctionality). Then, in Section 4.3, we show to t-privately compute the F_{mult} functionality for any $t < n/2$. This involves specifying and implementing two subfunctionalities F_{rand}^{2t} and F_{reduce}^{deg}; see the beginning of Section 4.3 for an overview of the protocol for t-privately computing F_{mult} and for the definition of these subfunctionalities.

4.2 Private Computation in the F_{mult}-Hybrid Model

In this section we present a formal description and proof of the protocol for t-privately computing any deterministic functionality f in the F_{mult}-hybrid model. As we have mentioned, it is assumed that all inputs are in a known field $\mathbb{F}$ of size greater than n, and that the arithmetic circuit C is over $\mathbb{F}$. Clearly, the field $\mathbb{F}$ and thus the circuit C are determined by the lengths of the inputs and the number of parties. For simplicity, we also assume that each party's input and output is a single field element (and thus is associated with a single input and output wire). Thus, we refer to a functionality of the form $f : \mathbb{F}^n \to \mathbb{F}^n$. The extension to the case where a party's input may be comprised of multiple field elements is straightforward.

PROTOCOL 4.1 (t-Private Computation in the F_{mult}-Hybrid Model)

- **Inputs:** Each party P_i holds private input $x_i \in \mathbb{F}$
- **Auxiliary input:** Each party P_i holds an arithmetic circuit C over the field $\mathbb{F}$, such that for every $\vec{x} \in \mathbb{F}^n$ it holds that $C(\vec{x}) = f(\vec{x})$, where $f : \mathbb{F}^n \to \mathbb{F}^n$. The parties also hold a description of $\mathbb{F}$ and distinct non-zero values $\alpha_1, \ldots, \alpha_n$ in $\mathbb{F}$.
- **The protocol:**
  1. **The input stage:** For every $i \in \{1, \ldots, n\}$, upon input x_i, party P_i chooses a polynomial $q_i(x)$ uniformly from the set $\mathcal{P}^{x_i,t}$. For every $j \in \{1, \ldots, n\}$, P_i sends party P_j the value $q_i(\alpha_j)$.

     Each party P_i records the values $q_1(\alpha_i), \ldots, q_n(\alpha_i)$ that it received.
  2. **The computation stage:** Let $g_1, \ldots, g_\ell$ be a predetermined topological ordering of the gates of the circuit. For $k = 1, \ldots, \ell$ the parties work as follows:
     - *Case 1 – g_k is an addition gate:* Let β_k^i and γ_k^i be the shares of input wires held by party P_i. Then, P_i defines its share of the output wire to be $\delta_k^i = \beta_k^i + \gamma_k^i$.
     - *Case 2 – g_k is a multiplication-by-a-constant gate with constant c:* Let β_k^i be the share of the input wire held by party P_i. Then, P_i defines its share of the output wire to be $\delta_k^i = c \cdot \beta_k^i$.
     - *Case 3 – g_k is a multiplication gate:* Let β_k^i and γ_k^i be the shares of input wires held by party P_i. Then, P_i sends (β_k^i, γ_k^i) to the ideal functionality F_{mult} and receives back a value δ_k^i. Party P_i defines its share of the output wire to be δ_k^i.
  3. **The output stage:** Let $o_1, \ldots, o_n$ be the output wires, where party P_i's output is the value on wire o_i. For every $k = 1, \ldots, n$, denote by $\beta_k^1, \ldots, \beta_k^n$ the shares that the parties hold for wire o_k. Then, each P_i sends P_k its share β_k^i.

     Upon receiving all shares, P_k computes $\mathsf{reconstruct}(\beta_k^1, \ldots, \beta_k^{t+1})$ and obtains a polynomial $g_k(x)$ (note that $t+1$ of the n shares suffice). P_k then defines its output to be $g_k(0)$.

We now prove the security of Protocol 4.1. We remark that in the F_{mult}-hybrid model, the protocol is actually t-private for *any* $t < n$. However, as we will see, in order to t-privately compute the F_{mult} functionality, we will need to set $t < n/2$.

Theorem 4.2 *Let $\mathbb{F}$ be a finite field, let $f : \mathbb{F}^n \to \mathbb{F}^n$ be an n-ary functionality, and let $t < n$. Then, Protocol 4.1 t-privately computes f in the F_{mult}-hybrid model, in the presence of a static semi-honest adversary.*

Proof: Intuitively, the protocol is t-private because the only values that the parties see until the output stage are random shares. Since the threshold of the secret sharing scheme used is $t + 1$, it holds that no adversary controlling t parties can learn anything. The fact that the view of the adversary can be simulated is due to the fact that t shares of any two secrets are identically distributed, as was formally proven in Claim 3.2. This implies that the simulator can generate the shares based on any arbitrary value, and the resulting view is identical to that of a real execution. Observe that this is true until the output stage where the simulator must make the random shares that were used match the actual output of the corrupted parties. This is not a problem because, by interpolation, any set of t shares can be used to define a t-degree polynomial with its free coefficient being the actual output.

Since C computes the functionality f, it is immediate that $\text{OUTPUT}^{\pi}(x_1,\ldots,x_n) = f(x_1,\ldots,x_n)$, where π denotes Protocol 4.1. We now proceed to show the existence of a simulator $\mathcal{S}$ as required by Definition 2.2. Before describing the simulator, we present some necessary notation. Our proof works by inductively showing that the partial view of the adversary at every stage is identical in the simulated and real executions. Recall that the view of party P_i is the vector $(x_i, r_i; m_1^i, \ldots, m_\ell^i)$, where x_i is the party's input, r_i its random tape, m_k^i is the kth message that it receives in the execution, and ℓ is the overall number of messages received (in our context here, we let m_k^i equal the series of messages that P_i receives when the parties compute gate g_k). For the sake of clarity, we add to the view of each party the values $\sigma_1^i, \ldots, \sigma_\ell^i$, where σ_k^i equals the shares on the wires that Party P_i holds *after* the parties compute gate g_k. That is, we denote

$$\text{VIEW}_i^{\pi}(\vec{x}) = \left(x_i, r_i; m_1^i, \sigma_1^i, \ldots, m_\ell^i, \sigma_\ell^i\right).$$

We stress that since the σ_k^i values can be efficiently computed from the party's input, random tape and incoming messages, this is equivalent and only a matter of notation.

Denote by $\text{VIEW}_I^{\pi,0}(\vec{x})$ the VIEW of the corrupted parties after the input stage. Furthermore, for every $k \in \{1,\ldots,\ell\}$, denote by $\text{VIEW}_I^{\pi,k}(\vec{x})$ the view of the parties after the execution of gate g_k. Finally, denote by $\text{VIEW}_I^{\pi,\ell+1}(\vec{x})$ the view of the parties at the end of the protocol; that is, $\text{VIEW}_I^{\pi,\ell+1}(\vec{x}) = \text{VIEW}_I^{\pi}(\vec{x})$. Equivalently, we denote by $\mathcal{S}^0\left(I, \vec{x}_I, f_I\left(\vec{x}\right)\right)$ the view generated by the simulator after the input stage, by $\mathcal{S}^k\left(I, \vec{x}_I, f_I\left(\vec{x}\right)\right)$ the view generated by the simulator after simulating the execution of the kth gate, and by $\mathcal{S}^{\ell+1}\left(I, \vec{x}_I, f_I\left(\vec{x}\right)\right)$ the final output $\mathcal{S}(I, \vec{x}_I, f_I(\vec{x}))$ of the simulator.

We are now ready to describe the simulator $\mathcal{S}$. Loosely speaking, $\mathcal{S}$ works by simply sending random shares of arbitrary values until the output stage. Then, in the final output stage $\mathcal{S}$ sends values so that the reconstruction of the shares on the output wires yield the actual output.

The Simulator $\mathcal{S}$:

- **Input:** *The simulator receives the inputs and outputs, $\{x_i\}_{i\in I}$ and $\{y_i\}_{i\in I}$ respectively, of all corrupted parties.*
- **Simulation:**
  - Simulating the input stage:
    1. *For every $i \in I$, $\mathcal{S}$ chooses a uniformly distributed random tape for P_i; this random tape and the input x_i fully determines the degree-t polynomial $q_i'(x) \in \mathcal{P}^{x_i,t}$ chosen by P_i in the protocol.*
    2. *For every $j \notin I$, $\mathcal{S}$ chooses a random degree-t polynomial $q_k'(x) \in_R \mathcal{P}^{0,t}$ with free coefficient 0.*
    3. *The view of the corrupted parties is then recorded by $\mathcal{S}$ to be $\{q_1'(\alpha_i), \ldots, q_n'(\alpha_i)\}_{i\in I}$ along with the random tapes chosen for P_i, for every $i \in I$.*

– Simulating the computation stage: *For every $g_k \in \{g_1, \ldots, g_\ell\}$:*

1. g_k is an addition gate: *Let $\{f(\alpha_i)\}_{i\in I}$ and $\{g(\alpha_i)\}_{i\in I}$ be the shares of the input wires of the corrupted parties that were generated by $\mathcal{S}$ (initially these are input wires and so the shares are defined by $q'_k(x)$ above). For every $i \in I$, the simulator $\mathcal{S}$ computes $f(\alpha_i) + g(\alpha_i) = (f + g)(\alpha_i)$ which defines the shares of the output wire of g_k.*
2. g_k is a multiplication-with-constant gate: *Let $\{f(\alpha_i)\}_{i\in I}$ be the shares of the input wire and let $c \in \mathbb{F}$ be the constant of the gate. $\mathcal{S}$ computes $c \cdot f(\alpha_i) = (c \cdot f)(\alpha_i)$ for every $i \in I$ which defines the shares of the output wire of g_k.*
3. g_k is a multiplication gate: *$\mathcal{S}$ chooses a degree-t polynomial $f(x)$ uniformly at random from $\mathcal{P}^{0,t}$ (irrespective of the shares of the input wires), and defines the shares of the corrupted parties of the output wire of g_k to be $\{f(\alpha_i)\}_{i\in I}$.*

$\mathcal{S}$ adds the shares to the corrupted parties' views.

– Simulating the output stage: *Let $o_1, \ldots, o_n$ be the output wires. We now focus on the output wires of the corrupted parties. For every $k \in I$, the simulator $\mathcal{S}$ has already defined $|I|$ shares $\{\beta^i_k\}_{i\in I}$ for the output wire o_k. $\mathcal{S}$ thus chooses a* random polynomial *$g'_k(x)$ of degree t under the following constraints:*

1. *$g'_k(0) = y_k$, where y_k is the corrupted P_k's output (the polynomial's free coefficient is the correct output).*
2. *For every $i \in I$, $g'_k(\alpha_i) = \beta^i_k$ (i.e., the polynomial is consistent with the shares that have already been defined).*

(Note that if $|I| = t$, then the above constraints fully defines $t + 1$ points, which in turn fully defines the polynomial $g'_k(x)$. However, if $|I| < t$, then $\mathcal{S}$ can carry out the above by choosing $t - |I|$ additional random points and interpolating.)

Finally, $\mathcal{S}$ adds the shares $\{g'_k(\alpha_1), \ldots, g'_k(\alpha_n)\}$ to the view of the corrupted party P_k.

– *$\mathcal{S}$ outputs the views of the corrupted parties and halts.*

We now show by induction that the partial view (of any length) of the corrupted parties in a real execution is identically distributed to the partial view (of the same length) output by the simulator. We start with the base case, after the input stage has concluded.

Claim 4.3 *For every $\vec{x} \in \mathbb{F}^n$ and every $I \subset [n]$ with $|I| \leq t$,*

$$\left\{\text{VIEW}^{\pi,0}_I(\vec{x})\right\} \equiv \left\{\mathcal{S}^0\left(I, \vec{x}_I, f_I\left(\vec{x}\right)\right)\right\}$$

Proof: In a real execution, the corrupted parties receive shares of all the real inputs $\vec{x}$, whereas in the simulation they receive shares of $\vec{x}'$ (recall that $x'_i = x_i$ for every $i \in I$, and $x'_j = 0$ for every $j \notin I$). The view of the corrupted parties in a real execution equals:

$$\text{VIEW}^{\pi,0}_I(\vec{x}) = \left\{ \{(q_1(\alpha_i), \ldots, q_n(\alpha_i))\}_{i\in I}\right\}$$

where $q_k(x)$ is chosen uniformly at random in $\mathcal{P}^{x_k,t}$ for every $k \in [n]$. The view of the corrupted parties in the simulation equals:

$$\mathcal{S}^0\left(I, \vec{x}_I, f_I\left(\vec{x}\right)\right) = \left\{ \{(q'_1(\alpha_i), \ldots, q'_n(\alpha_i))\}_{i\in I}\right\}$$

where for every $i \in I$ we have $q'_i(x) \in_R \mathcal{P}^{x_i,t}$, and for every $j \notin I$ we have $q'_j(x) \in_R \mathcal{P}^{0,t}$. Thus, these differ on the polynomials $q'_j(x)$ for all $j \notin I$. Nevertheless, by Claim 3.4 (using $m = n - |I|$), we have that

$$\left\{ \{q_j(\alpha_i)\}_{j \notin I; i \in I} \right\} \equiv \left\{ \{q'_j(\alpha_i)\}_{j \notin I; i \in I} \right\}.$$

Thus,

$$\left\{ \{(q_1(\alpha_i), \ldots, q_n(\alpha_i))\}_{i \in I} \right\} \equiv \left\{ \{(q'_1(\alpha_i), \ldots, q'_n(\alpha_i))\}_{i \in I} \right\}$$

and the real and simulated views of the input stage are identical. ■

We now prove the inductive step:

Claim 4.4 *Let $k \in \{1, \ldots, \ell\}$. If*

$$\left\{ \text{VIEW}_I^{\pi,k-1}(\vec{x}) \right\} \equiv \left\{ \mathcal{S}^{k-1}(I, \vec{x}_I, f_I(\vec{x})) \right\},$$

then

$$\left\{ \text{VIEW}_I^{\pi,k}(\vec{x}) \right\} \equiv \left\{ \mathcal{S}^{k}(I, \vec{x}_I, f_I(\vec{x})) \right\}.$$

Proof: Let $k \in \{1, \ldots, \ell\}$. Then the gate being computed is g_k; we separately consider the case that g_k is an addition or multiplication-by-constant gate and the case that g_k is a multiplication gate.

If g_k is an addition or multiplication-by-constant gate, then $\text{VIEW}_I^{\pi,k}(\vec{x})$ is a *deterministic function* of $\text{VIEW}_I^{\pi,k-1}(\vec{x})$, and $\mathcal{S}^k(I, \vec{x}_I, f_I(\vec{x}))$ is a deterministic function of $\mathcal{S}^{k-1}(I, \vec{x}_I, f_I(\vec{x}))$. This is due to the fact that the extension of the view involves either adding shares that appear in the partial view of length $k-1$ or multiplying them by a constant. Thus, if the partial views of length $k-1$ are identically distributed it is immediate that the partial views of length k are identically distributed. (Stated differently, if X and Y are identically distributed, then $A(X)$ and $A(Y)$ are identically distributed for any algorithm A. In this case, A can be defined as the partial-view extension operation described above.)

If g_k is a multiplication gate, then the parties all send their shares on their input wires to F_{mult} who sends back random shares of the actual value $a \cdot b$ of the output wire. In contrast, the simulator chooses these values as random shares of zero. Now, by Claim 3.2 t random shares of $a \cdot b$ are identically distributed to t random shares of 0. In addition, by the assumption, the partial views until this point are also identically distributed. Thus, if the shares added in this step are *independent* of the partial view of length $k-1$, then the partial views of length k are also identically distributed. (This holds because if $X_1 \equiv Y_1$ and $X_2 \equiv Y_2$, and in addition X_1, X_2 are independent and Y_1, Y_2 are independent, then $(X_1, X_2) \equiv (Y_1, Y_2)$.) However, both F_{mult} and the simulator $\mathcal{S}$ choose the shares randomly and independently of everything else that was generated so far. Thus, independence holds as required. (We remark that the fact that in the real execution the shares are to $a \cdot b$ which is related to the previous values is actually of no consequence. This is because Claim 3.2 holds when both values being shared (in this case $a \cdot b$ and 0) are known, and likewise the fact that the partial views of length $k-1$ are identically distributed holds when $\vec{x}$ is known; this latter fact is implicit in the fact that $\vec{x}$ is fixed in the ensemble. Now, observe that the value $a \cdot b$ is *fully determined* given $\vec{x}$ and thus any dependence of the partial view on this value would contradict the inductive assumption regarding the partial view of length $k-1$.) This completes the proof of the claim. ■

It remains to show that the output of the simulation after the output stage is identical to the view of the corrupted parties in a real execution. For simplicity, we assume that the output wires appear immediately after multiplication gates (otherwise, they are fixed functions of these values).

Before proving this, we prove a lemma that describes the processes of the real execution and simulation in a more abstract way. The aim of the lemma is to prove that the process carried out by the simulator in the output stage yields the same distribution as in a protocol execution. We first describe two processes and prove that they yield the same distribution, and later show how these are actually the real and simulation processes.

Random Variable $X(s)$	**Random Variable $Y(s)$**
(1) Choose $f(x) \in_R \mathcal{P}^{s,t}$	(1) Choose $f'(x) \in_R \mathcal{P}^{0,t}$
(2) $\forall i \in I$, set $\beta_i = f(\alpha_i)$	(2) $\forall i \in I$, set $\beta'_i = f'(\alpha_i)$
(3) –	(3) Choose $g(x) \in_R \mathcal{P}^{s,t}$ s.t. $\forall i \in I$ $g(\alpha_i) = \beta'_i$
(4) Output $f(x)$	(4) Output $g(x)$

Observe that in $Y(s)$ the polynomial $f'(x)$ is first chosen with free coefficient 0, and then only afterwards $g(x)$ is chosen with free coefficient s.

Lemma 4.5 *For every $s \in \mathbb{F}$, it holds that $\{X(s)\} \equiv \{Y(s)\}$.*

Proof: Define $X(s) = (X_1, X_2)$ and $Y(s) = (Y_1, Y_2)$, where X_1 (resp., Y_1) are the output values from step (2) of the process, and X_2 (resp., Y_2) are the output polynomials from step (4) of the process. From Claim 3.2, it immediately follows that $\{X_1\} \equiv \{Y_1\}$. It therefore suffices to show that $\{X_2 \mid X_1\} \equiv \{Y_2 \mid Y_1\}$. Stated equivalently, we wish to show that for every set of field values $\{\beta_i\}_{i \in I}$ and every $h(x) \in \mathcal{P}^{s,t}$,

$$\Pr\left[X_2(x) = h(x) \mid \forall i : X_2(\alpha_i) = \beta_i\right] = \Pr\left[Y_2(x) = h(x) \mid \forall i : Y_2(\alpha_i) = \beta_i\right]$$

where $\{\beta_i\}_{i \in I}$ are the conditioned values in X_1 and Y_1 (we use the same β_i for both since these are identically distributed and we are now conditioning them). First, if there exists an $i \in I$ such that $h(\alpha_i) \neq \beta_i$ then both probabilities above are 0. We now compute these probabilities for the case that $h(\alpha_i) = \beta_i$ for all $i \in I$. We first claim that

$$\Pr\left[Y_2(x) = h(x) \mid \forall i : Y_2(\alpha_i) = \beta_i\right] = \frac{1}{|\mathbb{F}|^{t-|I|}}.$$

This follows immediately from the process for generating the random variable $Y(s)$, because $Y_2(x) = g(x)$ is chosen at random under the constraint that for every $i \in I$, $g(\alpha_i) = \beta_i$. Since $|I| + 1$ points are already fixed (the β_i values and the free coefficient s), there are $|\mathbb{F}|^{t-|I|}$ different polynomials of degree-t that meet these constraints, and Y_2 is chosen uniformly from them.

It remains to show that

$$\Pr\left[X_2(x) = h(x) \mid \forall i : X_2(\alpha_i) = \beta_i\right] = \frac{1}{|\mathbb{F}|^{t-|I|}}.$$

In order to see this, observe that

$$\Pr\left[X_2(x) = h(x) \ \wedge \ \forall i : X_2(\alpha_i) = \beta_i\right] = \Pr\left[X_2(x) = h(x)\right]$$

because in this case, we consider only polynomials $h(x)$ for which $h(\alpha_i) = \beta_i$ for all $i \in I$, and so the conditioning adds nothing. We conclude that

$$\begin{aligned}\Pr\left[X_2(x) = h(x) \mid \forall i : X_2(\alpha_i) = \beta_i\right] &= \frac{\Pr\left[X_2(x) = h(x) \ \wedge \ \forall i : X_2(\alpha_i) = \beta_i\right]}{\Pr\left[\forall i : X_2(\alpha_i) = \beta_i\right]} \\ &= \frac{\Pr\left[X_2(x) = h(x)\right]}{\Pr\left[\forall i : X_2(\alpha_i) = \beta_i\right]} \\ &= \frac{1}{|\mathbb{F}|^t} \cdot |\mathbb{F}|^{|I|} = \frac{1}{|\mathbb{F}|^{t-|I|}},\end{aligned}$$

where the last equality follows because $f(x) = X_2(x) \in_R \mathcal{P}^{s,t}$ and by Claim 3.2 the probability that the points $X_2(\alpha_i) = \beta_i$ for all $i \in I$ equals $|\mathbb{F}|^{-|I|}$. ∎

The random variables $X(s)$ and $Y(s)$ can be extended to $X(\vec{s})$ and $Y(\vec{s})$ for any $\vec{s} \in \mathbb{F}^m$ (for some $m \in \mathbb{N}$); the proof of the analogous lemma then follows from standard arguments, as in Corollary 3.4.

Claim 4.6 *If*

$$\left\{\text{VIEW}_I^{\pi,\ell}(\vec{x})\right\} \equiv \left\{\mathcal{S}^{\ell}\left(I, \vec{x}_I, f_I\left(\vec{x}\right)\right)\right\}, \tag{3}$$

then

$$\left\{\text{VIEW}_I^{\pi,\ell+1}(\vec{x})\right\} \equiv \left\{\mathcal{S}^{\ell+1}\left(I, \vec{x}_I, f_I\left(\vec{x}\right)\right)\right\}.$$

Proof: In the output stage, for every $k \in I$, the corrupted parties receive the points $g_k(\alpha_1), \ldots, g_k(\alpha_n)$ (resp. the points $g'_k(\alpha_1), \ldots, g'_k(\alpha_n)$ in the simulation). Equivalently, we can say that the corrupted parties receive the polynomials $\{g_k(x)\}_{k\in I}$ (resp. $\{g'_k(x)\}_{k\in I}$).

In the protocol execution, functionality F_{mult} chose the polynomial $f_k(x)$ for the output wire of P_k uniformly at random in $\mathcal{P}^{y_k,t}$. Then, the corrupted parties received values $\beta_i = f_k(\alpha_i)$ (for every $i \in I$). Finally, as we have just described, in the output stage, the corrupted parties receive the polynomials $f_k(x)$ themselves. Thus, this is *exactly* the process $X(y_k)$. Extending to all $k \in I$, we have that this is the extended process $X(\vec{s})$ with $\vec{s}$ being the vector containing the corrupted parties' output values $\{y_k\}_{k\in I}$.

In contrast, in the simulation of the multiplication gate leading to the output wire for party P_k, the simulator $\mathcal{S}$ chose the polynomial $f'_k(x)$ uniformly at random in $\mathcal{P}^{0,t}$ (see the specification of $\mathcal{S}$ above), and the corrupted parties received values $\beta_i = f_k(\alpha_i)$ (for every $i \in I$). Now, in the output stage, $\mathcal{S}$ chose $g'_k(x)$ at random from $\mathcal{P}^{y_k,t}$ under the constraint that $g'_k(\alpha_i) = \beta_i$ for every $i \in I$. Thus, this is *exactly* the process $Y(y_k)$. Extending to all $k \in I$, we have that this is the extended process $Y(\vec{s})$ with $\vec{s}$ being the vector containing the corrupted parties' output values $\{y_k\}_{k\in I}$. The claim thus follows from Lemma 4.5. ∎

Combining Claims 4.3 to 4.6 we have that $\left\{\mathcal{S}^{\ell+1}\left(I, \vec{x}_I, f_I\left(\vec{x}\right)\right)\right\} \equiv \left\{\text{VIEW}_I^{\pi,\ell+1}(\vec{x})\right\}$. This completes the proof because $\mathcal{S}^{\ell+1}\left(I, \vec{x}_I, f_I\left(\vec{x}\right)\right) = \mathcal{S}\left(I, \vec{x}_I, f_I(\vec{x})\right)$ and $\text{VIEW}_I^{\pi,\ell+1}(\vec{x}) = \text{VIEW}_I^{\pi}\left(\vec{x}\right)$.

∎

Privately computing linear functionalities in the real model. Theorem 4.2 states that every function can be t-privately computed in the F_{mult}-hybrid model, for *any* $t < n$. However, a look at Protocol 4.1 and its proof of security show that F_{mult} is only used for computing multiplication gates in the circuit. Thus, Protocol 4.1 can actually be directly used for privately computing any linear functionality f, since such functionalities can be computed by circuits containing only addition and multiplication-by-constant gates. Furthermore, the protocol is secure for any $t < n$; in particular, no honest majority is needed. This yields the following corollary.

Corollary 4.7 *Let $t < n$. Then, any linear functionality f can be t-privately computed in the presence of a static semi-honest adversary. In particular, the matrix-multiplication functionality $F_A(\vec{x}) = A \cdot \vec{x}$ for matrix $A \in \mathbb{F}^{n\times n}$ can be t-privately computed in the presence of a static semi-honest adversary.*

4.3 Privately Computing the F_{mult} Functionality

We have shown how to t-privately compute any functionality in the F_{mult}-hybrid model. In order to achieve private computation in the plain model, it therefore remains to show how to privately compute the F_{mult} functionality. We remark that the threshold needed to privately compute F_{mult} is $t < n/2$, and thus the overall threshold for the generic BGW protocol is $t < n/2$. Recall that the F_{mult} functionality is defined by

$$F_{mult}\Big((f_a(\alpha_1), f_b(\alpha_1)), \ldots, (f_a(\alpha_n), f_b(\alpha_n))\Big) = \Big(f_{ab}(\alpha_1), \ldots, f_{ab}(\alpha_n)\Big)$$

where $f_a(x) \in \mathcal{P}^{a,t}$, $f_b(x) \in \mathcal{P}^{b,t}$, and $f_{ab}(x) \in_R \mathcal{P}^{a\cdot b,t}$ for some a, b. Equivalently, this can defined as follows:

$$F_{mult}((\beta_1, \gamma_1), \ldots, (\beta_n, \gamma_n)) = (\delta_1, \ldots, \delta_n)$$

where $\mathsf{reconstruct}(\beta_1, \ldots, \beta_n)$ is a polynomial $f_a(x) \in \mathcal{P}^{a,t}$, $\mathsf{reconstruct}(\gamma_1, \ldots, \gamma_n)$ is a polynomial $f_b(x) \in \mathcal{P}^{b,t}$, and $\mathsf{reconstruct}(\delta_1, \ldots, \delta_n)$ is a *random* polynomial $f_{ab}(x) \in_R \mathcal{P}^{a\cdot b,t}$. (Here we use $\mathsf{reconstruct}$ to denote the unique polynomial defined by the points it receives.)

As we have discussed above, the simple solution where each party locally multiplies its two shares does not work here, for two reasons. First, the resulting polynomial is of degree $2t$ and not t as required. Second, the resulting polynomial is not uniformly distributed amongst all polynomials with the required free coefficient. Based on the above, the F_{mult} functionality is computed according to the following steps:

1. Each party locally multiplies its input shares.

2. The parties run a protocol to generate a random polynomial in $\mathcal{P}^{0,2t}$, and each party receives a share based on this polynomial. Then, each party adds its share of the product (from the previous step) with its share of this polynomial. The resulting shares thus define a polynomial which is uniformly distributed in $\mathcal{P}^{a\cdot b,2t}$.

3. The parties run a protocol to reduce the degree of the polynomial to t, with the result being a polynomial that is uniformly distributed in $\mathcal{P}^{a\cdot b,t}$, as required. This computation uses a t-private protocol for computing *matrix multiplication*. Fortunately, we have already shown how to achieve this in Corollary 4.7 at the end of Section 4.2.

The randomizing and degree-reduction functionalities for carrying out the above steps are formally defined as follows:

- *The randomization functionality:* The randomization functionality is defined as follows:

$$F_{rand}^{2t}(\lambda, \ldots, \lambda) = (\sigma_1, \ldots, \sigma_n),$$

where $\mathsf{reconstruct}(\sigma_1, \ldots, \sigma_n) \in_R \mathcal{P}^{0,2t}$, and λ denotes the empty string. We will show how to t-privately compute this functionality in Section 4.3.2.

- *The degree-reduction functionality:* Let $h(x) = h[0] + \ldots + h[2t]x^{2t}$ be a polynomial, and denote by $\mathsf{trunc}_t(h(x))$ the polynomial of degree t with coefficients $h[0], \ldots, h[t]$. That is, $\mathsf{trunc}_t(h(x)) = h[0]+h[1]x+\ldots+h[t]x^t$. (Observe that this is a deterministic functionality.) Formally, we define $F_{reduce}^{deg}(\beta_1, \ldots, \beta_n) = (\delta_1, \ldots, \delta_n)$ where $\mathsf{reconstruct}(\delta_1, \ldots, \delta_n) = \mathsf{trunc}_t(\mathsf{reconstruct}(\beta_1, \ldots, \beta_n))$. Note that if $\mathsf{reconstruct}(\beta_1, \ldots, \beta_n)$ is a random polynomial then it has random coefficients. This then implies that the truncated polynomial is also random (because it has the same coefficients). We will show how to t-privately compute this functionality in Section 4.3.3.

4.3.1 Privately Computing F_{mult} in the $(F_{rand}^{2t}, F_{reduce}^{deg})$-Hybrid Model

We now prove that F_{mult} is reducible to the functionalities F_{rand}^{2t} and F_{reduce}^{deg}; that is, we construct a protocol that t-privately computes F_{mult} given access to ideal functionalities F_{reduce}^{deg} and F_{rand}^{2t}.

PROTOCOL 4.8 (t-Privately Computing F_{mult})

- **Input:** Each party P_i holds values β_i, γ_i, such that $\mathsf{reconstruct}(\beta_1, \ldots, \beta_n) \in \mathcal{P}^{a,t}$ and $\mathsf{reconstruct}(\gamma_1, \ldots, \gamma_n) \in \mathcal{P}^{b,t}$ for some $a, b \in \mathbb{F}$.
- **The protocol:**
  1. Each party locally computes $s_i = \beta_i \cdot \gamma_i$.
  2. **Randomize:** Each party P_i sends λ to F_{rand}^{2t} (formally, it writes λ on its oracle tape for F_{rand}^{2t}). Let σ_i be the oracle response for party P_i.
  3. **Reduce the degree:** Each party P_i sends $(s_i + \sigma_i)$ to F_{reduce}^{deg}. Let δ_i be the oracle response for P_i.
  4. Each party P_i outputs δ_i.

Proposition 4.9 *Let $t < n/2$. Then, Protocol 4.8 t-privately computes F_{mult} in the $(F_{rand}^{2t}, F_{reduce}^{deg})$-hybrid model, in the presence of a static semi-honest adversary.*

Proof: The parties do not receive messages from other parties in the oracle-aided protocol; rather they receive messages from the oracles only. Therefore, our simulator only needs to simulate the oracle-response messages. Since the F_{mult} functionality is probabilistic, we must prove its security using Definition 2.1.

In the real execution of the protocol, the corrupted parties' inputs are given by $\{f_a(\alpha_i)\}_{i\in I}$ and $\{f_b(\alpha_i)\}_{i\in I}$. Then, in the randomize step of the protocol they receive shares σ_i of a random polynomial of degree $2t$ with free coefficient 0. Denoting this polynomial by $r(x)$, we have that the corrupted parties receive the values $\{r(\alpha_i)\}_{i\in I}$. Next, the parties invoke the functionality F_{reduce}^{deg}

and receive back the values δ_i (these are points of the polynomial $\mathsf{trunc}_t(f_a(x) \cdot f_b(x) + r(x))$). These values are actually the parties' output, and thus the simulator must make the output of the call to F^{deg}_{reduce} be the shares $\{\delta_i\}_{i \in I}$ of the corrupted parties outputs.

The simulator $\mathcal{S}$:

- **Input:** *The simulator receives as input I, the inputs of the corrupted parties $\{(\beta_i, \gamma_i)\}_{i \in I}$, and their outputs $\{\delta_i\}_{i \in I}$.*
- **Simulation:**
  - *$\mathcal{S}$ chooses $|I|$ values uniformly and independently at random, $\{v_i\}_{i \in I}$.*
  - *For every $i \in I$, the simulator defines the view of the party P_i to be: $(\beta_i, \gamma_i, v_i, \delta_i)$, where (β_i, γ_i) is P_i's input, v_i is P_i's oracle response from F^{2t}_{rand}, and δ_i is P_i's oracle response from F^{deg}_{reduce}.*

Intuitively, the above simulation strategy is as required since F^{2t}_{rand} returns shares that are independent of both the input and output. It is immediate that the shares are independent of the input. Regarding the output, this follows from the fact that F^{2t}_{rand} returns a random polynomial of degree $2t$ and the output shares are influenced only by the first t coefficients. Thus, the last t coefficients provide sufficient randomness to create independence between the $|I| \leq t$ shares viewed by the corrupted parties after F^{2t}_{rand}, and the overall output of the protocol.

We now proceed to prove that the joint distribution of the output of all the parties, together with the view of the corrupted parties is distributed identically to the output of all parties as computed from the functionality F_{mult} and the output of the simulator. We first show that the outputs of all parties are distributed identically in both cases. Then, we show that the view of the corrupted parties is distributed identically, given the outputs (and inputs) of all parties.

The outputs. Since the inputs and outputs of all the parties lie on the same polynomials, it is enough to show that the polynomials are distributed identically. Let $f_a(x), f_b(x)$ be the input polynomials. Let $r(x)$ be the output of the F^{2t}_{rand} functionality. Finally, denote the truncated result by $\hat{h}(x) \stackrel{\text{def}}{=} \mathsf{trunc}(f_a(x) \cdot f_b(x) + r(x))$.

In the real execution of the protocol, the parties output shares based on the polynomial $\hat{h}(x)$. From the way it is defined, it is immediate that $\hat{h}(x)$ is a degree-t polynomial that is uniformly distributed in $\mathcal{P}^{a \cdot b, t}$. (In order to see that it is uniformly distributed, observe that with the exception of the free coefficient, all the coefficients of the degree-$2t$ polynomial $f_a(x) \cdot f_b(x) + r(x)$ are random. Thus the coefficients of $x, \ldots, x^t$ in $\hat{h}(x)$ are random, as required.)

On the other hand, the functionality F_{mult} return shares for a random polynomial of degree t that hides the free coefficient $f_a(0) \cdot f_b(0) = a \cdot b$. Thus, the outputs of the parties in both cases are distributed identically.

The view of the corrupted parties. We show that the view of the corrupted parties is distributed identically, given the inputs and outputs of all parties. Observe that the inputs and outputs define the polynomials $f_a(x)$, $f_b(x)$ and $f_{ab}(x)$. Now, the output of the simulator is

$$\Big\{ \{f_a(\alpha_i), f_b(\alpha_i), v_i, f_{ab}(\alpha_i)\}_{i \in I} \Big\}$$

where all the v_i values are uniformly distributed in $\mathbb{F}$, and independent of $f_a(x), f_b(x)$ and $f_{ab}(x)$. It remains to show that in a protocol execution the analogous values – which are the outputs received by the corrupted parties from F_{rand}^{2t} – are also uniformly distributed and independent of $f_a(x), f_b(x)$ and $\hat{h}(x)$ (where $\hat{h}(x)$ is distributed identically to a random $f_{ab}(x)$, as already shown above). In order to prove this, it suffices to prove that for every vector $\vec{y} \in \mathbb{F}^{|I|}$,

$$\Pr\left[\vec{y} = \vec{r} \mid f_a(x), f_b(x), \hat{h}(x)\right] = \frac{1}{|\mathbb{F}|^{|I|}} \tag{4}$$

where $\vec{r} = (r(\alpha_{i_1}), \ldots, r(\alpha_{i_{|I|}}))$ for $I = \{i_1, \ldots, i_{|I|}\}$; i.e., $\vec{r}$ is the vector of outputs from F_{rand}^{2t}, computed from the polynomial $r(x) \in_R \mathcal{P}^{0,2t}$, that are received by the corrupted parties. Observe that it suffices to prove Eq. (4) since if this probability is indeed $1/|\mathbb{F}|^{|I|}$, then this implies that $\vec{r}$ is comprised of $|I|$ uniformly and independently distributed elements from $\mathbb{F}$.

We write $r(x) = r_1(x) + x^t \cdot r_2(x)$, where $r_1(x) \in_R \mathcal{P}^{0,t}$ and $r_2(x) \in_R \mathcal{P}^{0,t}$. In addition, we write $f_a(x) \cdot f_b(x) = h_1(x) + x^t \cdot h_2(x)$, where $h_1(x) \in \mathcal{P}^{ab,t}$ and $h_2(x) \in \mathcal{P}^{0,t}$. Observe that:

$$\hat{h}(x) = \mathsf{trunc}\left(f_a(x) \cdot f_b(x) + r(x)\right) = \mathsf{trunc}\left(h_1(x) + r_1(x) + x^t \cdot (h_2(x) + r_2(x))\right) = h_1(x) + r_1(x)$$

where the last step is true since the free coefficient of both $h_2(x), r_2(x)$ is 0, and they therefore add nothing to the coefficient of x^t. Rewriting Eq. (4), we need to prove that for every vector $\vec{y} \in \mathbb{F}^{|I|}$,

$$\Pr\left[\vec{y} = \vec{r} \mid f_a(x), f_b(x), h_1(x) + r_1(x)\right] = \frac{1}{|\mathbb{F}|^{|I|}}$$

where the kth element r_k of $\vec{r}$ is $r_1(\alpha_{i_k}) + (\alpha_{i_k})^t \cdot r_2(\alpha_{i_k})$. It follows that for any given $y_k \in \mathbb{F}$, the equality $y_k = r_1(\alpha_{i_k}) + (\alpha_{i_k})^t \cdot r_2(\alpha_{i_k})$ holds if and only if $r_2(\alpha_{i_k}) = (\alpha_{i_k})^{-t} \cdot (y_k - r_1(\alpha_{i_k}))$, where by the conditioning all the values $\alpha_{i_k}, y_k, r_1(\alpha_{i_k})$ on the right hand side are *fixed* (observe that $h_1(x)$ is fixed given $f_a(x)$ and $f_b(x)$, and thus $r_1(x)$ is fixed given $f_a(x), f_b(x)$ and $h_1(x) + r_1(x)$). In addition, by the protocol description, $r_2(x)$ is clearly independent of $f_a(x), f_b(x)$ and $h_1(x) + r_1(x)$. We conclude that Eq. (4) holds if and only if

$$\Pr\left[\bigwedge_{k \in I} r_2(\alpha_{i_k}) = (\alpha_{i_k})^{-t} \cdot (y_k - r_1(\alpha_{i_k}))\right] = \frac{1}{|\mathbb{F}|^{|I|}},$$

which is exactly what is proven in Corollary 3.3. We conclude that the view of the corrupted parties is identically distributed to the output of the simulator, when conditioning on the inputs and outputs of all parties. ■

4.3.2 Privately Computing F_{rand}^{2t} in the Plain Model

The randomization functionality is defined as follows:

$$F_{rand}^{2t}(\lambda, \ldots, \lambda) = (\delta_1, \ldots, \delta_n),$$

where $\mathsf{reconstruct}(\delta_1, \ldots, \delta_n) \in_R \mathcal{P}^{0,2t}$, and λ denotes the empty string. The protocol for implementing the functionality works as follows. Each party P_i chooses a random polynomial $q_i(x) \in_R \mathcal{P}^{0,2t}$ and sends the share $q_i(\alpha_j)$ to every party P_j. Then, each party P_i outputs $\delta_i = \sum_{k=1}^{n} q_k(\alpha_i)$.

Clearly, the shares $\delta_1, \ldots, \delta_n$ define a polynomial with free coefficient 0, because all the polynomials in the sum have a zero free coefficient. Furthermore, the sum of random $2t$-degree polynomials is a random polynomial in $\mathcal{P}^{0,2t}$, as required. See Protocol 4.10 for a formal description.

PROTOCOL 4.10 (Privately Computing F^{2t}_{rand})

- **Input:** The parties do not have inputs for this protocol.
- **The protocol:**
  - Each party P_i chooses a random polynomial $q_i(x) \in_R \mathcal{P}^{0,2t}$. Then, for every $j \in \{1, \ldots, n\}$ it sends $s_{i,j} = q_i(\alpha_j)$ to party P_j.
  - Each party P_i receives $s_{1,i}, \ldots, s_{n,i}$ and computes $\delta_i = \sum_{j=1}^{n} s_{j,i}$.
  - Each party P_i outputs δ_i.

We now prove that Protocol 4.10 t-privately computes F^{2t}_{rand}.

Claim 4.11 *Let $t < n/2$. Then, Protocol 4.10 t-privately computes the F^{2t}_{rand} functionality, in the presence of a static semi-honest adversary.*

Proof: Intuitively, the protocol is secure because the only messages that the parties receive are random shares of polynomials in $\mathcal{P}^{0,2t}$. The simulator can easily simulate these messages by generating the shares itself. However, in order to make sure that the view of the corrupted parties is consistent with the actual output provided by the functionality, the simulator needs to choose the shares so that their sum equals δ_i, the output provided by the functionality to each P_i.

The simulator $\mathcal{S}$:

- **Input:** *The simulator receives as input I and the outputs of the corrupted parties $\{\delta_i\}_{i \in I}$.*
- **Simulation:**
  - *$\mathcal{S}$ chooses n random polynomials $q'_j(x) \in_R \mathcal{P}^{0,2t}$ for every $j \in [n]$, under the constraint that $(\sum_{j=1}^{n} q'_j(\alpha_i)) = \delta_i$, for every $i \in I$. (Note that for $i \in I$, this involves setting the random tape of P_i so that it results in it choosing $q'_i(x)$.)*
  - *$\mathcal{S}$ sets $s'_{j,i} = q'_j(\alpha_i)$ for every $j \in [n]$ and every $i \in I$, and writes the incoming messages of corrupted party P_i in the protocol to be $(s'_{1,i}, \ldots, s'_{n,i})$.*

We now show that the view of the corrupted parties and the output of all parties is distributed identically to the output of the simulator and the output of all parties as received from the functionality. The output of all the parties lie on a single polynomial and so we show that the joint distribution of the corrupted parties' view (containing the polynomials $\{q_i(x)\}_{i \in I}$ and the points $\{q_j(\alpha_i)\}_{j \notin I; i \in I}$) and the output polynomial is the same in the real and ideal executions.

More formally, we need to prove that:

$$\left\{ \{q_i(x), q_1(\alpha_i), \ldots, q_n(\alpha_i)\}_{i \in I}, \sum_{j=1}^{n} q_j(x) \right\} \equiv \left\{ \{q'_i(x), q'_1(\alpha_i), \ldots, q'_n(\alpha_i)\}_{i \in I}, q'(x) \right\} \tag{5}$$

where for every $j \in [n]$ the polynomials $q_j(x)$ are distributed uniformly at random in $\mathcal{P}^{0,2t}$ (this is the left-hand side that relates to a real protocol execution), and the polynomials $q'_j(x)$ are distributed uniformly at random in $\mathcal{P}^{0,2t}$ under the constraint that $\sum_{j=1}^{n} q'_j(\alpha_i) = \delta_i$, for every $i \in I$. The polynomial $q'(x)$ is the unique polynomial passing through all points $\{(\alpha_j, \delta_j)\}_{j=1}^{n}$.

First observe that since all the polynomials $q'(x), q_1(x), \ldots, q_n(x)$ are chosen uniformly at random from $\mathcal{P}^{0,2t}$, we have that

$$\left\{ q'(x) \right\} \equiv \left\{ \sum_{j=1}^{n} q_j(x) \right\},$$

implying that the outputs of all parties are identically distributed in the real and ideal execution. We proceed to show that the view of the corrupted parties is distributed identically, conditioned on the output of all parties. That is, we show that:

$$\left\{ \{q_i(x), q_1(\alpha_i), \ldots, q_n(\alpha_i)\}_{i\in I} \;\Big|\; \sum_{j=1}^{n} q_j(x) \right\} \equiv \left\{ \{q'_i(x), q'_1(\alpha_i), \ldots, q'_n(\alpha_i)\}_{i\in I} \;\Big|\; q'(x) \right\}$$

The conditional distribution on the left-hand side is equivalent to fixing a polynomial $q(x)$, choosing $q_1(x), \ldots, q_n(x)$ at random under the constraint that $\sum_{j=1}^{n} q_j(x) = q(x)$, and then outputting the corrupted parties' polynomials $\{q_i(x)\}_{i\in I}$ and points $\{q_1(\alpha_i), \ldots, q_n(\alpha_i)\}_{i\in I}$. In contrast, the conditional distribution on the right-hand side is exactly what the simulator does (since it receives points from an already fixed $q'(x)$). The actions of the simulator are equivalent to choosing the points $q'_1(\alpha_i), \ldots, q'_n(\alpha_i)$ at random under the constraint that $\sum_{j=1}^{n} q'_j(\alpha_i) = q'(\alpha_i) = \delta_i$, for every $i \in I$, and then choosing the remaining points completely at random. The equivalence follows since these two processes are identical (choosing random polynomials under the constraint and outputting points and choosing random points under the constraint and completing the polynomials at random). ■

4.3.3 Privately Computing F^{deg}_{reduce} in the $F_A(\vec{x})$-Hybrid Model

Recall that the F^{deg}_{reduce} functionality is defined by

$$F^{deg}_{reduce}(\beta_1, \ldots, \beta_n) = (\delta_1, \ldots, \delta_n),$$

where $\mathsf{reconstruct}(\delta_1, \ldots, \delta_n) = \mathsf{trunc}_t(\mathsf{reconstruct}(\beta_1, \ldots, \beta_n))$, and $\mathsf{trunc}_t(h(x))$ is the polynomial of degree t with coefficients $h[0], \ldots, h[t]$. We begin by showing that in order to transform a vector of shares of the polynomial $h(x)$ to shares of the polynomial $\mathsf{trunc}_t(h(x))$, it suffices to multiply the input shares by a certain matrix of constants.

Claim 4.12 *There exists a constant matrix $A \in \mathbb{F}^{n\times n}$ such that the following holds. For every pair of vectors $\vec{\beta} = (\beta_1, \ldots, \beta_n)$ and $\vec{\delta} = (\delta_1, \ldots, \delta_n)$ such that* $\mathsf{reconstruct}(\beta_1, \ldots, \beta_n)$ *is a polynomial of degree $2t$ with $t < n/2$ and* $\mathsf{reconstruct}(\delta_1, \ldots, \delta_n) = \mathsf{trunc}_t(\mathsf{reconstruct}(\beta_1, \ldots, \beta_n))$*, it holds that $\vec{\delta} = A \cdot \vec{\beta}$.*

Proof: Let $h(x) = \mathsf{reconstruct}(\beta_1, \ldots, \beta_n)$ and denote $h(x) = h[0] + h[1]x + \ldots h[2t]x^{2t}$. Furthermore, denote by $\vec{h}$ the vector of length n:

$$\vec{h} = (h[0], \ldots, h[t], \ldots, h[2t], 0, \ldots 0)\ .$$

By definition, $\beta_i = h(\alpha_i)$ for every $i \in \{1,\ldots,n\}$. Let $V_{\vec{\alpha}}$ be the $n \times n$ Vandermonde matrix for the set $\vec{\alpha}$. As we have seen in Section 3.5, it holds that:

$$V_{\vec{\alpha}} \cdot \vec{h} = \vec{\beta} \; .$$

Since $V_{\vec{\alpha}}$ is invertible, we have that

$$\vec{h} = V_{\vec{\alpha}}^{-1} \cdot \vec{\beta}. \tag{6}$$

Let $k = \mathsf{trunc}_t(h)$ be the truncated polynomial $k(x) = h[0] + h[1]x + \ldots h[t]x^t$. Again, by definition, $\delta_i = k(\alpha_i)$. Furthermore, denoting

$$\vec{k} = (h[0], \ldots, h[t], 0, \ldots, 0).$$

we have that

$$V_{\vec{\alpha}} \cdot \vec{k} = \vec{\delta} \; . \tag{7}$$

Now, let $T = \{1,\ldots,t\}$, and let P_T be the linear projection of T; i.e., P_T is an $n \times n$ matrix such that $P_T(i,j) = 1$ for every $i = j \in T$, and $P_T(i,j) = 0$ for all other values. It thus follows that

$$P_T \cdot \vec{h} = \vec{k} \; . \tag{8}$$

Combining Equations (7) and (8), we have that:

$$V_{\vec{\alpha}} \cdot P_T \cdot \vec{h} = \vec{\delta} \; .$$

Next, combining this with Eq. (6) we have:

$$V_{\vec{\alpha}} \cdot P_T \cdot V_{\vec{\alpha}}^{-1} \cdot \vec{\beta} = \vec{\delta}.$$

Defining the constant matrix $A \stackrel{\text{def}}{=} V_{\vec{\alpha}} \cdot P_T \cdot V_{\vec{\alpha}}^{-1}$, we have that for every $\vec{\beta}$ and $\vec{\delta}$ as in the claim, it holds that

$$A \cdot \vec{\beta} = \vec{\delta},$$

as required. ■

By the above claim it follows that the parties can compute F_{reduce}^{deg} by simply multiplying their shares with the constant matrix A from above. That is, the entire protocol for t-privately computing F_{reduce}^{deg} works by the parties t-privately computing the matrix multiplication functionality $F_A(\vec{x})$ with matrix A as above. By Corollary 4.7 (see the end of Section 4.2), $F_A(\vec{x})$ can be t-privately computed for any $t < n$. Since the entire degree reduction procedure consists of t-privately computing $F_A(\vec{x})$, we have the following proposition:

Proposition 4.13 *For every $t < n/2$, there exists a protocol for t-privately computing the F_{reduce}^{deg}, in the presence of a static semi-honest adversary.*

4.4 Conclusion

In Section 4.3.1 we proved that there exists a t-private protocol for computing the F_{mult} functionality in the $(F_{rand}^{2t}, F_{reduce}^{deg})$-hybrid model, for any $t < n/2$. Then, in Sections 4.3.2 and 4.3.3 we showed that F_{rand}^{2t} and F_{reduce}^{deg}, respectively, can be t-privately computed (in the plain model) for any $t < n/2$. Finally, in Theorem 4.2 we showed that any n-ary functionality can be privately computed in the F_{mult}-hybrid model, for any $t < n$. Combining the above with the modular sequential composition theorem described in Section 2.3, we conclude that:

Corollary 4.14 *Let $\mathbb{F}$ be a finite field, let $f : \mathbb{F}^n \to \mathbb{F}^n$ be an n-ary functionality, and let $t < n/2$. Then, there exists a protocol for t-privately computing f in the presence of a static semi-honest adversary.*

5 Verifiable Secret Sharing (VSS)

5.1 Background

Verifiable secret sharing [12] (VSS) is a protocol for sharing a secret in the presence of malicious adversaries. Recall that a secret sharing scheme (with threshold $t+1$) is made up of two stages. In the first stage, the dealer shares a well-defined secret so that any $t+1$ parties can later reconstruct the secret, while any subset of t or fewer parties will learn nothing whatsoever about the secret. In the second stage, a set of $t+1$ or more parties reconstruct the secret. If we consider Shamir's secret-sharing scheme, as in the semi-honest BGW construction, much can go wrong if the adversary is malicious. First, in order to share a secret s, the dealer is supposed to choose a random polynomial $q(\cdot)$ of degree t with $q(0) = s$ and then hand each party P_i its share $q(\alpha_i)$. However, nothing prevents the dealer from choosing a polynomial of higher degree. This is a problem because it means that different subsets of $t+1$ parties may reconstruct different values. Thus, the shared value is not well defined. Second, in the reconstruction phase each party P_i provides its share $q(\alpha_i)$. However, a corrupted party can provide a different value, thus effectively changing the value of the reconstructed secret, and the other parties have no way of knowing that the provided value is incorrect. Thus, we must use a method that either prevents the corrupted parties from presenting incorrect shares, or ensures that it is possible to reconstruct the correct secret s given $n-t$ correct shares even if they are mixed together with t incorrect shares (and no one knows which of the shares are correct or incorrect). Note that in the context of multiparty computation, n parties participate in the reconstruction and not just $t+1$; this is utilized in the construction.

The BGW protocol for verifiable secret sharing solves the above problems by adding elements to the share stage of the protocol so that the shares received by the honest parties (with $t < n/3$) are guaranteed to be $q(\alpha_i)$ for a well-defined degree-t polynomial q, even if the dealer is corrupted. Then, they show that as long as $t < n/3$, it is possible to use techniques from the field of error-correcting codes in order to reconstruct q (and thus $q(0) = s$) as long as $n-t$ correct shares are provided. Indeed, they observe that Shamir's secret-sharing scheme when looked at in this context is exactly a Reed-Solomon code, and Reed-Solomon codes can correct up to t errors, for $t < n/3$.

5.2 The Reed-Solomon Code

We briefly describe the Reed-Solomon code, and its use in our context. First, recall that a linear $[n, k, d]$-code over a field $\mathbb{F}$ of size q is a code of length n (meaning that each codeword is comprised of n field elements), of dimension k (meaning that there are q^k different codewords), and of distance d (meaning that every two codewords are of Hamming distance at least d from each other).

We are interested in constructing a code of length n and dimension $k = t+1$. The Reed-Solomon code for these parameters is constructed as follows. Let $\mathbb{F}$ be a finite field such that $\mathbb{F} > n$, and let $\alpha_1, \ldots, \alpha_n$ be distinct field elements. Let $m = (m_0, \ldots, m_t)$ be a message to be encoded, where each $m_i \in \mathbb{F}$. The encoding of m is as follows:

1. Define a polynomial $p_m(x) = m_0 + m_1 x + \ldots + m_t x^t$ of degree t
2. Compute the codeword $C(m) = \langle p_m(\alpha_1), \ldots, p_m(\alpha_n)\rangle$

It is well known that the distance of this code is $n - t$. (In order to see this, recall that for any two different polynomials p_1, p_2 of degree at most t, there are at most t points α for which $p_1(\alpha) = p_2(\alpha)$. Noting that $m \neq m'$ define different polynomials $p_m \neq p_{m'}$, we have that $C(m)$ and $C(m')$ agree in at most t places, and so $d(C(m), C(m')) \geq n - t$.) The following is a well-known fact from the error correcting code literature:

Fact 5.1 *The Reed-Solomon code is a linear $[n, t + 1, n - t]$-code. In addition, there exists an efficient decoding algorithm that corrects up to $\frac{n-t-1}{2}$ errors. That is, for every $m \in \mathbb{F}^{t+1}$ and every $x \in \mathbb{F}^n$ such that $d(x, C(m)) \leq \frac{n-t-1}{2}$, the decoding algorithm returns m.*

Let $t < n/3$, and so $n \geq 3t + 1$. Plugging this into the above, we have that it is possible to efficiently correct up to $\frac{3t+1-t-1}{2} = t$ errors.

Reed-Solomon and Shamir's secret-sharing. Assume that n parties hold shares $q(\alpha_i)$ of a degree-t polynomial, as in Shamir's secret-sharing scheme. That is, the dealer distributed shares $q(\alpha_i)$ where $q \in_R \mathcal{P}^{s,t}$ for a secret $s \in \mathbb{F}$. We can view the shares $\langle q(\alpha_1), \ldots, q(\alpha_n)\rangle$ as a Reed-Solomon codeword. Now, in the reconstruction phase, the honest parties all provide their correct share $q(\alpha_i)$, whereas the corrupted parties may provide incorrect values. However, since the number of corrupted parties is $t < n/3$, it follows that at most t of the symbols are incorrect.[3] Thus, the Reed-Solomon reconstruction procedure can be run and the honest parties can all obtain the correct polynomial q, and can compute $q(0) = s$.

We conclude that in such a case the corrupted parties cannot effectively cheat in the reconstruction phase. Indeed, even if they provide incorrect values, it is possible for the honest parties to correctly reconstruct the secret *with probability* 1. Thus, the main challenge in constructing a verifiable secret-sharing protocol is how to force a corrupted dealer to distribute shares that are consistent with a degree-t polynomial. Once we have achieved this, the corrupted parties cannot do anything to interfere with the reconstruction of the correct secret.

[3]This is one of the reasons that in the case of a malicious adversary, we can only tolerate $t < n/3$ corrupted parties, whereas in the case of a semi-honest adversary we could tolerate up to $t < n/2$ corrupted parties.

5.3 Bivariate Polynomials

Bivariate polynomials are a central tool used by the BGW verifiable secret sharing protocol (in the sharing stage). We therefore provide a short background to bivariate polynomials in this section.

A bivariate polynomial of degree t is a polynomial over two variables, each of which has degree at most t. Such a polynomial is defined as follows:

$$f(x,y) = \sum_{i=0}^{t} \sum_{j=0}^{t} a_{i,j} \cdot x^i \cdot y^j.$$

We denote by $\mathcal{B}^{s,t}$ the set of all bivariate polynomials of degree t and with free coefficient s (we stress that whenever we say the degree t of a bivariate polynomial, we mean that it is of degree *at most* t in *each one of the variables*). Note that the number of coefficients of a bivariate polynomial in $\mathcal{B}^{s,t}$ is $(t+1)^2 - 1 = t^2 + 2t$ (there are $(t+1)^2$ coefficients, but the free coefficient is already fixed to be s).

When considering univariate polynomials, $t+1$ points define a unique polynomial of degree t. In this case, each point is a pair (α_k, β_k) and there exists a unique polynomial f such that $f(\alpha_k) = \beta_k$ for all $t+1$ given points $\{(\alpha_k, \beta_k)\}_{k=1}^{t+1}$.

The analogous statement for bivariate polynomials is that $t+1$ univariate polynomials of degree t define a unique bivariate polynomial of degree t. In the case of a degree-t bivariate polynomial $S(x,y)$, fixing the y-value to be some α defines a degree-t univariate polynomial $f(x) = S(x,\alpha)$. Likewise, fixing any $t+1$ values $\alpha_1,\ldots,\alpha_{t+1}$ fully defines $t+1$ degree-t univariate polynomials $f_k(x) = S(x,\alpha_k)$. What we show now is that like in the univariate case, this works in the opposite direction as well. Specifically, given $t+1$ values $\alpha_1,\ldots,\alpha_{t+1}$ and $t+1$ degree-t polynomials $f_1(x),\ldots,f_{t+1}(x)$ there exists a unique bivariate polynomial $S(x,y)$ such that $S(x,\alpha_k) = f_k(x)$, for every $k = 1,\ldots,t+1$. This is formalized in the next claim, which was proven in [15]:

Claim 5.2 *Let t be a nonnegative integer, let $\alpha_1,\ldots,\alpha_{t+1}$ be $t+1$ distinct elements in $\mathbb{F}$, and let $f_1(x),\ldots,f_{t+1}(x)$ be $t+1$ polynomials of degree t. Then, there exists a unique bivariate polynomial $S(x,y)$ of degree t such that for every $k = 1,\ldots,t+1$ it holds that:*

$$S(x,\alpha_k) = f_k(x) \tag{9}$$

Proof: Define the bivariate polynomial $S(x,y)$ via the Lagrange interpolation:

$$S(x,y) = \sum_{i=1}^{t+1} f_i(x) \cdot \frac{\prod_{j\neq i}(y-\alpha_j)}{\prod_{j\neq i}(\alpha_i-\alpha_j)}$$

It is easy to see that $S(x,y)$ has degree t. Moreover, for every $k = 1,\ldots,t+1$ it holds that:

$$\begin{aligned}
S(x,\alpha_k) &= \sum_{i=1}^{t+1} f_i(x) \cdot \frac{\prod_{j\neq i}(\alpha_k-\alpha_j)}{\prod_{j\neq i}(\alpha_i-\alpha_j)} \\
&= f_k(x) \cdot \frac{\prod_{j\neq k}(\alpha_k-\alpha_j)}{\prod_{j\neq k}(\alpha_k-\alpha_j)} + \sum_{i\in[t+1]\setminus\{k\}} f_i(x) \cdot \frac{\prod_{j\neq i}(\alpha_k-\alpha_j)}{\prod_{j\neq i}(\alpha_i-\alpha_j)} \\
&= f_k(x) + 0 \\
&= f_k(x)
\end{aligned}$$

and $S(x, y)$ therefore satisfies Eq. (9). It remains to show that S is unique. Assume that there exist two different t-degree bivariate polynomials $S_1(x, y)$ and $S_2(x, y)$ that satisfy Eq. (9). Define the polynomial

$$R(x, y) \overset{\text{def}}{=} S_1(x, y) - S_2(x, y) = \sum_{i=0}^{t} \sum_{j=0}^{t} r_{i,j} x^i y^j .$$

We will now show that $R(x, y) = 0$. First, for every $k \in [t + 1]$ it holds that:

$$R(x, \alpha_k) = \sum_{i,j=0}^{t} r_{i,j} x^i (\alpha_k)^j = S_1(x, \alpha_k) - S_2(x, \alpha_k) = f_k(x) - f_k(x) = 0,$$

where the last equality follows from Eq. (9). We can rewrite the univariate polynomial $R(x, \alpha_k)$ as

$$R(x, \alpha_k) = \sum_{i=0}^{t} \left(\left(\sum_{j=0}^{t} r_{i,j} (\alpha_k)^j \right) \cdot x^i \right) .$$

As we have seen, $R(x, \alpha_k) = 0$ for every x. Thus, its coefficients are all zeroes,[4] implying that for every fixed $i \in [t + 1]$ it holds that $\sum_{j=0}^{t} r_{i,j} (\alpha_k)^j = 0$. This in turn implies that for every fixed $i \in [t+1]$, the polynomial $h_i(x) = \sum_{j=0}^{t} r_{i,j} x^j$ is zero for $t+1$ points (i.e., the points $\alpha_1, \ldots, \alpha_{t+1}$), and so $h_i(x)$ is also the zero polynomial. Thus, its coefficients $r_{i,j}$ equal 0 for every $j \in [t+1]$. This holds for every fixed i, and therefore for every $i, j \in [t+1]$ we have that $r_{i,j} = 0$. We conclude that $R(x, y) = 0$ for every x and y, and hence $S_1(x, y) = S_2(x, y)$. ■

Secret sharing from bivariate polynomials. It is possible to define a secret sharing scheme using bivariate polynomials as follows. Let $s \in \mathbb{F}$ be a secret, let $\alpha_1, \ldots, \alpha_n$ be distinct non-zero field elements, and let $S(x, y) \in \mathcal{B}^{s,t}$ be a random degree-t polynomial with free coefficient s. Then, party P_i's share is the univariate polynomial $f_i(x) = S(x, \alpha_i)$. Observe that by Claim 5.2, any $t+1$ parties can reconstruct S given their shares. In addition to the above, we define P_i's "dual share" to be the univariate polynomial $g_i(y) = S(\alpha_i, y)$. In the protocol for verifiable secret sharing, each party P_i will receive $f_i(x)$ and $g_i(y)$. This enables P_i to check the shares received by the other parties. This is due to the fact that for every $i, j \in [n]$, it holds that $f_i(\alpha_j) = S(\alpha_j, \alpha_i) = g_j(\alpha_i)$, and $g_i(\alpha_j) = S(\alpha_i, \alpha_j) = f_j(\alpha_i)$.

The VSS protocol works by embedding a univariate degree-t polynomial $q(z)$ with $q(0) = s$ into the bivariate polynomial $S(x, y)$. Specifically, $S(x, y)$ is chosen at random under the constraint that $S(0, z) = q(z)$; the values $q(\alpha_1), \ldots, q(\alpha_n)$ are thus the univariate Shamir-shares embedded into $S(x, y)$. Observe that based on the above definition of the polynomials $f_i(x)$ from $S(x, y)$, we have that $f_i(0) = S(0, \alpha_i) = q(\alpha_i)$. Thus, given the bivariate share $(f_i(x), g_i(y))$ party P_i can locally compute its univariate Shamir-share as $q(\alpha_i) = f_i(0)$.

We now prove a "secrecy lemma" based on bivariate polynomials. Loosely speaking, we prove that the shares $\{f_i(x), g_i(y)\}_{i \in I}$ that the corrupted parties receive do not reveal any information about the secret s. In fact, we show something much stronger: for every two degree-t polynomials

[4]In order to see that all the coefficients of a polynomial which is identical to zero are zeroes, let $p(x) = \sum_{i=0}^{t} a_i x^t$, where $p(x) = 0$ for every x. Let $\vec{a}$ be a vector of the coefficients of p, and let $\vec{\beta}$ be some vector of size $t + 1$ of some distinct non-zero elements. Let $V_{\vec{\beta}}$ be the Vandermonde matrix for $\vec{\beta}$. Then, $V_{\vec{\beta}} \cdot \vec{a} = 0$, and therefore $\vec{a} = V_{\vec{\beta}}^{-1} \cdot 0 = 0$.

q_1, q_2 such that $q_1(\alpha_i) = q_2(\alpha_i) = f_i(0)$, the distribution over the shares $\{f_i(x), g_i(y)\}_{i\in I}$ received by the corrupted parties when $S(x,y)$ is chosen based on $q_1(z)$ is identical to the distribution when $S(x,y)$ is chosen based on $q_2(z)$. An immediate corollary of this is that no information is revealed about the secret $s_1 = q_1(0)$ or $s_2 = q_2(0)$. Note that the following is essentially the bivariate analogue of Claim 3.2.

Claim 5.3 *Let $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ be n distinct non-zero values. For every $I \subset [n]$ where $|I| \leq t$ and every two degree-t polynomials q_1, q_2 over $\mathbb{F}$ for which $q_1(\alpha_i) = q_2(\alpha_i)$ for every $i \in I$, it holds that:*

$$\left\{\{(i, f_i^{(1)}(x), g_i^{(1)}(y))\}_{i\in I}\right\} \equiv \left\{\{(i, f_i^{(2)}(x), g_i^{(2)}(y))\}_{i\in I}\right\}$$

where $S_1(x,y) \in_R \mathcal{B}^{q_1(0),t}$ under the constraint that $S_1(0,z) = q_1(z)$ and $S_2(x,y) \in_R \mathcal{B}^{q_2(0),t}$ under the constraint that $S_2(0,z) = q_2(z)$, and $f_i^{(1)}(x) = S_1(x,\alpha_i)$, $g_i^{(2)}(y) = S_1(\alpha_i, y)$, $f_i^{(2)}(x) = S_2(x,\alpha_i)$ and $g_i^{(2)}(y) = S_2(\alpha_i, y)$, for every $i \in I$.

Proof: We begin by defining ensembles $\mathbb{S}_1$ and $\mathbb{S}_2$, as follows:

$$\begin{aligned}
\mathbb{S}_1 &= \left\{\{(i, S_1(x,\alpha_i), S_1(\alpha_i, y))\}_{i\in I} \mid S_1 \in_R \mathcal{B}^{q_1(0),t} \text{ s.t. } S_1(0,z) = q_1(z)\right\} \\
\mathbb{S}_2 &= \left\{\{(i, S_2(x,\alpha_i), S_2(\alpha_i, y))\}_{i\in I} \mid S_2 \in_R \mathcal{B}^{q_2(0),t} \text{ s.t. } S_2(0,z) = q_2(z)\right\}
\end{aligned}$$

Given this notation, an equivalent formulation of the claim is that $\mathbb{S}_1 \equiv \mathbb{S}_2$.

In order to prove that this holds, we first show that for any set of pairs of degree-t polynomials $Z = \{(i, f_i(x), g_i(y))\}_{i\in I}$, the number of bivariate polynomials in the support of $\mathbb{S}_1$ that are consistent with Z equals the number of bivariate polynomials in the support of $\mathbb{S}_2$ that are consistent with Z (where consistency means that $f_i(x) = S(x,\alpha_i)$ and $g_i(y) = S(y,\alpha_i)$). Now, if there exist $i,j \in I$ such that $f_i(\alpha_j) \neq g_j(\alpha_i)$ then there does not exist any bivariate polynomial in the support of $\mathbb{S}_1$ or $\mathbb{S}_2$ that is consistent with Z. In addition, if there exists an $i \in I$ such that $f_i(0) \neq q_1(\alpha_i)$, then once again there is no polynomial from $\mathbb{S}_1$ or $\mathbb{S}_2$ that is consistent. We stress that this holds for both $\mathbb{S}_1$ and $\mathbb{S}_2$ because $q_1(\alpha_i) = q_2(\alpha_i)$ for all $i \in I$.

Otherwise, there do exist, and we begin by counting how many such polynomials exist in the support of $\mathbb{S}_1$. We have that Z contains $|I|$ degree-t polynomials $\{f_i(x)\}_{i\in I}$, and recall that $t+1$ such polynomials $f_i(x)$ fully define a degree-t bivariate polynomial. Thus, we need to choose $t - |I| + 1$ more polynomials $f_j(x)$ ($j \neq i$) that are consistent with $q_1(z)$ and with $g_i(y)$. In order for a polynomial $f_j(x)$ to be consistent, it must hold that $f_j(\alpha_i) = g_i(\alpha_j)$ and in addition that $f_j(0) = q_1(\alpha_j)$. Thus, for each such $f_j(x)$ that we add, $|I|+1$ points of f_j are already determined. Since $t+1$ points determine a degree-t univariate polynomial, it follows that an additional $t - |I|$ points can be chosen in all possible ways and the result will be consistent with Z. We conclude that there exist $(t - |I| + 1) \cdot |\mathbb{F}|^{t-|I|}$ ways to choose S_1 according to $\mathbb{S}_1$ that will be consistent. (Note that if $|I| = t$ then there is just one way.) The important point here is that the exact same calculation holds for S_2 chosen according to $\mathbb{S}_2$, and thus exactly the same number of polynomials from $\mathbb{S}_1$ are consistent with Z as from $\mathbb{S}_2$.

Now, let $Z = \{(i, f_i(x), g_i(y))\}_{i\in I}$ be a set of $|I|$ pairs of univariate degree-t polynomials. We wish to show that the probability that Z is the set obtained from $\mathbb{S}_1$ equals the probability that Z is the set obtained from $\mathbb{S}_2$, where the probabilities are taken over the choice of S_1 and S_2 in $\mathbb{S}_1$ and $\mathbb{S}_2$, respectively. We have already seen that the number of polynomials in the support of $\mathbb{S}_1$ that are consistent with Z (0 or $(t-|I|+1)\cdot|\mathbb{F}|^{t-|I|}$) equals the number of polynomials in the support of $\mathbb{S}_2$ that are consistent with Z. In addition, the number of polynomials overall in the support of $\mathbb{S}_1$ equals the number of polynomials overall in the support of $\mathbb{S}_2$. Finally, the polynomials S_1 and S_2 are chosen *uniformly at randomly* from all polynomials in the support of $\mathbb{S}_1$ and $\mathbb{S}_2$, respectively. This implies that the probability that we obtain Z after choosing S_1 equals the probability that we obtain Z after choosing S_2. ■

5.4 The Verifiable Secret Sharing Protocol

In the VSS functionality, the dealer inputs a polynomial $q(x)$ of degree t, and the parties receive shares of that polynomial. The "verifiable" part is that if q is of degree greater than t, then the parties reject the dealer's shares and output $\perp$. The functionality is formally defined as follows:

$$F_{VSS}\left(q(x),\lambda,\ldots,\lambda\right)=\begin{cases}(q(\alpha_1),\ldots,q(\alpha_n)) & \text{if } \deg(q)\leq t\\ (\perp,\ldots,\perp) & \text{otherwise}\end{cases}$$

Observe that the secret $s=q(0)$ is only implicitly defined in the functionality; it is however well defined. Thus, in order to share a secret s, the functionality is used by having the dealer first choose a random polynomial $q\in_R\mathcal{P}^{s,t}$ and then running F_{VSS} with input $q(x)$.

We present the protocol of BGW with the simplification of the complaint phase suggested by [14]; this simplification was suggested in the BGW paper itself. The protocol uses private point-to-point channels between each pair of parties and an *authenticated* broadcast channel (meaning that the identity of the broadcaster is given); the broadcast channel can be implemented using Byzantine agreement [25, 21]. See Protocol 5.4 for a full specification.

The security of Protocol 5.4. Before we prove that Protocol 5.4 securely computes the F_{VSS} functionality, we present an intuitive argument as to why this holds. First, consider the case that the dealer is honest. In this case, an honest party P_j only broadcasts a complaint if a corrupted party sends it incorrect values. However, if this occurs then the dealer will not send a reveal of the honest party's polynomials (because its values are correct). Furthermore, if any corrupted party P_i broadcasts a complaint with incorrect values (u,v), the dealer can send the correct reveal message. In such a case, the check carried out by each honest party P_j in Step 5(b)ii will pass and so every honest party will broadcast good. Thus, at least $n-t$ parties broadcast good (since there are at least $n-t$ honest parties) and so all honest parties output $f_i(0)=S(0,\alpha_i)=q(\alpha_i)$, by the way the dealer chooses $S(x,y)$. Next, consider the case that the dealer is corrupted. In this case, the honest parties may receive polynomials that are not consistent with each other; i.e., P_j and P_k may receive polynomials $f_j(x),g_j(y)$ and $f_k(x),g_k(y)$ such that either $f_j(\alpha_k)\neq g_k(\alpha_j)$ or $f_k(\alpha_j)\neq g_j(\alpha_k)$. However, in such a case both parties complain, and the dealer must send a valid reveal message or no honest party will broadcast good. In order for $n-t$ parties to broadcast good, there must be at least $n-2t=t+1$ honest parties that broadcast good. This implies that these $t+1$ honest parties all received polynomials $f_j(x)$ and $g_j(y)$ that are consistent with all of the "fixed" values in the reveal messages. Thus, there are at least $t+1$ polynomials $f_j(x)$ that are consistent with all honest parties and these define a unique bivariate polynomial $S(x,y)$, as proven in Claim 5.2. We now proceed to the formal proof.

PROTOCOL 5.4 (Securely Computing F_{VSS})

- **Input:** The dealer $D = P_1$ holds a private polynomial $q(x)$ of degree at most t. The other parties $P_2, \ldots, P_n$ have no input.
- **Common input:** The description of a field $\mathbb{F}$ and n specified elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$.
- **The protocol:**
  1. **Round 1 (send shares) – the dealer:**
     (a) The dealer selects a uniformly distributed bivariate polynomial $S(x,y) \in_R \mathcal{B}^{q(0),t}$, under the constraint that $S(0,z) = q(z)$.
     (b) For every $i \in \{1, \ldots, n\}$, the dealer defines the polynomials $f_i(x) \stackrel{\text{def}}{=} S(x, \alpha_i)$ and $g_i(y) \stackrel{\text{def}}{=} S(\alpha_i, y)$. It then sends to each Party P_i the polynomials $f_i(x)$ and $g_i(y)$.
  2. **Round 2 (exchange subshares) – each party P_i:**
     (a) Store the polynomials $f_i(x)$ and $g_i(y)$ that were received from the dealer. (If $f_i(x)$ or $g_i(y)$ is of degree greater than t then truncate it to be of degree t.[5])
     (b) For every $j \in \{1, \ldots, n\}$, send $f_i(\alpha_j)$ and $g_i(\alpha_j)$ to party P_j.
  3. **Round 3 (broadcast complaints) – each party P_i:**
     (a) For every $j \in \{1, \ldots, n\}$, let (u_j, v_j) denote the values received from player P_j in Round 2 (these are supposed to be $u_j = f_j(\alpha_i)$ and $v_j = g_j(\alpha_i)$).
     If $u_j \neq g_i(\alpha_j)$ or $v_j \neq f_i(\alpha_j)$, then broadcast $\mathsf{complaint}(i, j, f_i(\alpha_j), g_i(\alpha_j))$.
     (b) If no parties broadcast a complaint, then every party P_i outputs $f_i(0)$ and halts.
  4. **Round 4 (resolve complaints) – the dealer:**
     (a) Upon viewing a message $\mathsf{complaint}(i, j, u, v)$ broadcast by P_i, check that $u = S(\alpha_j, \alpha_i)$ and $v = S(\alpha_i, \alpha_j)$. If the above condition holds, do nothing. Otherwise, broadcast $\mathsf{reveal}(i, f_i(x), g_i(y))$.
  5. **Round 5 (evaluate complaint resolutions) – each party P_i:**
     (a) If P_i views two messages $\mathsf{complaint}(k, j, u_1, v_1)$ and $\mathsf{complaint}(j, k, u_2, v_2)$ broadcast by P_k and P_j, respectively, such that $u_1 \neq v_2$ or $v_1 \neq u_2$, then P_i checks that the dealer broadcast the message $\mathsf{reveal}(k, f_k(x), g_k(y))$ or $\mathsf{reveal}(j, f_j(x), g_j(y))$. If the dealer did not broadcast either one of these messages then go to Step 6 (and do not broadcast good).
     (b) Upon receiving a message $\mathsf{reveal}(j, f_j(x), g_j(y))$, truncated to degree t if necessary as above:
        i. If $j = i$ then reset the stored polynomials $f_i(x)$ and $g_i(y)$ to the new polynomials that were received, and go to Step 6 (without broadcasting good).
        ii. If $j \neq i$ then check that $f_i(\alpha_j) = g_j(\alpha_i)$ and $g_i(\alpha_j) = f_j(\alpha_i)$. If the above condition does not hold, go to Step 6 (without broadcasting good).
     (c) Broadcast the message good.
  6. **Output decision – each party P_i:** If at least $n - t$ parties broadcast good, output $f_i(0)$. Otherwise, output $\perp$.

[5]If the parties do not truncate in this case, a corrupted dealer can send consistent polynomials of a higher degree to all parties, with the result that they will accept shares of a polynomial of degree greater than t. This check, that was not explicitly stated in [4], is therefore critical.

Theorem 5.5 *Let $t < n/3$. Then, Protocol 5.4 t-securely computes the F_{VSS} functionality in the presence of a static malicious adversary.*

Proof: Let $\mathcal{A}$ be an adversary in the real world. We show the existence of a simulator $\mathcal{S}$ such that for any set of corrupted parties I and for all inputs, the output of all parties and the adversary $\mathcal{A}$ in an execution of the real protocol with $\mathcal{A}$ is identical to the outputs in an execution with $\mathcal{S}$ in the ideal model. We separately prove the case that the dealer is honest and the case that the dealer is corrupted.

Case 1 – the Dealer is Honest

In this case in the ideal model, the dealer sends $q(x)$ to the trusted party and each honest party P_j outputs $q(\alpha_j)$, and never outputs $\perp$. Observe that none of the corrupted parties have input and so the adversary has no influence on the output of the honest parties. We begin by showing that this always holds in a real execution as well; i.e., in a real execution each honest party P_j always outputs $q(\alpha_j)$ and never outputs $\perp$.

Now, since the dealer is honest, it chooses a bivariate polynomial as described in the protocol and sends each party the prescribed values. In this case, an honest party P_j always outputs either $f_j(0)$ or $\perp$. This is due to the fact that its polynomial $f_j(x)$ will never be changed; it can only be changed if a $\mathsf{reveal}(j, f'_j(x), g_j(y))$ message is sent with $f'_j(x) \neq f_j(x)$. However, an honest dealer never does this. Thus, it remains to show that P_j never outputs $\perp$. In order to see this, recall that an honest party outputs $f_j(0) = q(\alpha_j)$ and not $\perp$ if and only if at least $n - t = 2t + 1$ parties broadcast good. Thus, it suffices to show that all honest parties broadcast good. An honest party P_j broadcasts good if and only if the following conditions hold:

- The dealer resolves all conflicts: whenever a pair of complaint messages $\mathsf{complaint}(k, \ell, u_1, v_1)$ and $\mathsf{complaint}(\ell, k, u_2, v_2)$ were broadcast such that $u_1 \neq v_2$ and $v_1 \neq u_2$ for some k and ℓ, the dealer broadcasts a reveal message for ℓ, k or both.
- Every revealed polynomial fits an honest P_j's polynomials: whenever the dealer broadcasts $\mathsf{reveal}(k, f_k(x), g_k(y))$, it holds that $f_j(\alpha_k) = g_k(\alpha_j)$ and $g_j(\alpha_k) = f_k(\alpha_j)$.
- The dealer did not broadcast $\mathsf{reveal}(j, f_j(x), g_j(y))$.

Since the dealer is honest, whenever there is a conflict between two parties, the dealer will broadcast a reveal message. This is due to the fact that if $u_1 \neq v_2$ or $u_2 \neq v_1$, it cannot hold that both (u_1, v_1) and (u_2, v_2) are consistent with $S(x, y)$. Thus, by its instructions, the dealer will broadcast at least one reveal message, and so condition (1) holds. In addition, it is immediate that since the dealer is honest, condition (2) also holds. Finally, the dealer broadcasts a $\mathsf{reveal}(j, f_j(x), g_j(y))$ message if and only if P_j sends a complaint with an *incorrect* pair (u, v); i.e., P_j broadcast (j, k, u, v) where $u \neq f_j(\alpha_k)$ or $v \neq g_j(\alpha_k)$. However, since both the dealer and P_j are honest, any complaint sent by P_j will be with the correct (u, v) values. Thus, the dealer will not broadcast a reveal of P_j's polynomials and condition (3) also holds. We conclude that every honest party broadcasts good and so all honest parties P_j output $f_j(0) = q(\alpha_j)$, as required.

Since the outputs of the honest parties are fully determined and always the same, it remains to show the existence of an ideal-model adversary/simulator $\mathcal{S}$ that can generate the *view of the adversary* $\mathcal{A}$ in an execution of the real protocol, given only the outputs $q(\alpha_i)$ of the corrupted parties P_i for every $i \in I$.

The simulator $\mathcal{S}$:

- *$\mathcal{S}$ invokes $\mathcal{A}$ on the auxiliary input z.*
- Interaction with the trusted party: *$\mathcal{S}$ receives the output values $\{q(\alpha_i)\}_{i\in I}$.*
- Generating the view of the corrupted parties: *$\mathcal{S}$ chooses any polynomial $q'(x)$ under the constraint that $q'(\alpha_i) = q(\alpha_i)$ for every $i \in I$. Then $\mathcal{S}$ runs all honest parties (including the honest dealer) in an interaction with $\mathcal{A}$, with the dealer input polynomial as $q'(x)$.*
- *$\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs, and halts.*

We now prove that the distribution generated by $\mathcal{S}$ is as required. We begin by observing that the only information that the corrupted parties view is the set of polynomials $\{f_i(x), g_i(y)\}_{i\in I}$. This set is what the dealer/simulator sends the corrupted parties. In addition, all points $f_j(\alpha_i), g_j(\alpha_i)$ sent by the honest parties can be computed by the adversary given the set $\{f_i(x), g_i(y)\}_{i\in I}$. Finally, the question of whether an honest party sends a complaint based on the corrupted parties' messages can be deterministically determined from the corrupted parties' messages alone. We stress that the simulation can be carried out given only this set of polynomials. This is due to the fact that even if an honest party complains, in the case of an honest dealer, the resolution of the complaint will involve broadcasting the corrupted party's polynomials only. Therefore, if the polynomials $\{f_i(x), g_i(y)\}_{i\in I}$ in the real protocol execution are identically distributed to those from the simulation (generated by the simulator from $q'(x)$), then the output distributions of the real and ideal execution are also identical. However, this is exactly what is proven in Claim 5.3. This completes the proof of the case that the dealer is honest.

Case 2 – the Dealer is Corrupted

In this case, the adversary $\mathcal{A}$ controls the dealer. Briefly speaking, the simulator $\mathcal{S}$ just plays the role of all honest parties (recall that all actions of the parties, apart from the dealer, are deterministic). If the simulated execution is such that the parties output $\perp$, the simulator sends an invalid polynomial (say $q(x) = x^{2t}$) to the trusted party. Otherwise, the simulator uses the fact that it sees all shares sent by $\mathcal{A}$ in order to interpolate and find the polynomial $q(x)$, which it then sends to the trusted party computing the functionality. We now formally describe the simulator:

The simulator $\mathcal{S}$:

1. *$\mathcal{S}$ invokes $\mathcal{A}$ on its auxiliary input z.*
2. *$\mathcal{S}$ plays the role of all the $n - |I|$ honest parties interacting with $\mathcal{A}$, as specified by the protocol, running until the end.*
3. *Let* num *be the number of (honest and corrupted) parties P_j that broadcast* good *in the simulation:*
   (a) *If* $\mathsf{num} < n - t$, *$\mathcal{S}$ sends the trusted party the polynomial $q(x) = x^{2t}$ as the dealer input.*
   (b) *If* $\mathsf{num} \geq n - t$, *then $\mathcal{S}$ defines a degree-t polynomial $q(x)$ as follows. For every honest party P_j ($j \notin I$) that broadcast* good *in the simulation, $\mathcal{S}$ defines the point $(\alpha_j, f_j(0))$. Then, $\mathcal{S}$ finds the degree-t polynomial $q'(x)$ that passes through all of these*

points $(\alpha_j, f_j(0))$. (In the analysis below we show that there is exactly one polynomial fulfilling this.) Finally, $\mathcal{S}$ sends $q'(x)$ to the trusted party (we stress that this is not necessarily related to the polynomial $q(x)$ that the dealer – equivalently P_1 – receives for input).

4. *$\mathcal{S}$ halts and outputs whatever $\mathcal{A}$ outputs.*

Observe that all parties, including the simulator, are *deterministic* since the only party who tosses coins in the protocol is the honest dealer (we can assume that $\mathcal{A}$ is deterministic because it is computationally unbounded). The concrete outputs of the parties are therefore fully determined; i.e., every time the protocol is run with $\mathcal{A}$ the exact same outputs are obtained, as is the case every time the simulation with $\mathcal{S}$ is run. We therefore show that the (concrete) outputs of the adversary and the parties in a real execution with $\mathcal{A}$ are equal to the outputs in an ideal execution with $\mathcal{S}$.

First, observe that the simulator plays the role of all the honest parties in an ideal execution, following the exact protocol specification. Since the honest parties have no input, the messages sent by the simulator in the ideal execution are exactly the same as those sent by the honest parties in a real execution of the protocol. Thus, the value that is output by $\mathcal{A}$ in a real execution *equals* the value that is output by $\mathcal{A}$ in the ideal execution with $\mathcal{S}$. It remains to show that the outputs of the honest parties are also the same in the real and ideal executions. Let OUTPUT_J denote the outputs of the parties P_j for all $j \in J$. We prove:

Claim 5.6 *Let $J = [n] \setminus I$ be the set of indices of the honest parties. For every adversary $\mathcal{A}$ controlling I including the dealer, every polynomial $q(x)$ and every auxiliary input $z \in \{0,1\}^*$ for $\mathcal{A}$, it holds that:*

$$\text{OUTPUT}_J\left(\text{REAL}_{\pi,\mathcal{A}(z),I}\left(q(x),\lambda,\ldots,\lambda\right)\right) = \text{OUTPUT}_J\left(\text{IDEAL}_{F_{VSS},\mathcal{S}(z),I}\left(q(x),\lambda,\ldots,\lambda\right)\right).$$

Proof: Let $\mathcal{A}$ be an adversary, and let $q(x)$ be the input polynomial of the dealer. Let $\vec{x} = (q(x),\lambda,\ldots,\lambda)$ (the vector of inputs). We separately analyze the case where some honest party outputs $\perp$ and the case where no honest party outputs $\perp$.

Case 1: There exists a $j \in J$ such that $\text{OUTPUT}_j(\text{REAL}_{\pi,\mathcal{A}(z),I}(q(x),\lambda,\ldots,\lambda)) = \perp$. We show that in this case all the honest parties output $\perp$ in both the real and ideal executions. Let j be such that $\text{OUTPUT}_j(\text{REAL}_{\pi,\mathcal{A}(z),I}(\vec{x})) = \perp$. By the protocol specification, an honest party P_j outputs $\perp$ (in the real world) if and only if it receives less than $n-t$ "good" messages over the broadcast channel. Since these messages are broadcast, it holds that all the parties receive the same messages. Thus, if an honest P_j output $\perp$ we have that it and all the honest parties received less than $n-t$ such "good" messages. Therefore, every honest party outputs $\perp$.

We now claim that in the ideal execution, all honest parties also output $\perp$. The output of the honest parties in the ideal execution are determined by the trusted third party, based on the input sent by $\mathcal{S}$. Thus, all honest parties output $\perp$ if and only if $\mathcal{S}$ sends x^{2t} to the trusted third party. As we have mentioned, the simulator $\mathcal{S}$ follows the instructions of the honest parties exactly in the simulation. Thus, if in a real execution with $\mathcal{A}$ it holds that less than $n-t$ parties broadcast good, then the same is also true in the simulation with $\mathcal{S}$. (We stress that *exactly the same messages* are sent by $\mathcal{A}$ and the honest parties in a real protocol execution and in the simulation with $\mathcal{S}$.) Now, by the instructions of $\mathcal{S}$, if less than $n-t$ parties broadcast good then num $< n-t$ and $\mathcal{S}$ sends $q(x) = x^{2t}$ to the trusted party. We conclude that all honest parties output $\perp$ in the ideal execution as well.

Case 2: For every $j \in J$ it holds that $\text{OUTPUT}_j(\text{REAL}_{\pi,\mathcal{A}(z),I}(\vec{x})) \neq \bot$. By what we have discussed above, this implies that in the simulation with $\mathcal{S}$, at least $n - t$ parties broadcast good. Since $n \geq 3t + 1$ this implies that at least $3t + 1 - t = 2t + 1$ parties broadcast good. Furthermore, since there are at most t corrupted parties, we have that at least $t + 1$ *honest* parties broadcast good. Recall that an honest party P_j broadcasts good if and only if the following conditions hold:

- The dealer resolves all conflicts.
- Every revealed polynomial fits P_j's polynomials.
- The dealer did not broadcast reveal$(j, f_j(x), g_j(y))$.

Let $K \subset [n]$ be a fixed subset of $t + 1$ honest parties that broadcast good (it is possible that more than $t + 1$ honest parties broadcast good; in such a case we take the $t + 1$ honest parties with the smallest indices). For each of these parties the above conditions hold. Thus, there exists a set of $t + 1$ pairs of polynomials $\{f_k(x), g_k(y)\}_{k \in K}$, such that for every $k, \ell \in K$ it holds that $f_k(\alpha_\ell) = g_\ell(\alpha_k)$. Taking the $t + 1$ distinct points $\{\alpha_k\}_{k \in K}$, we can apply Claim 5.2 that states that there exists a unique bivariate polynomial S such that $S(x, \alpha_k) = f_k(x)$ for every $k \in K$. The simulator $\mathcal{S}$ defines $q'(x)$ to be the unique polynomial that passes through all of the points $(\alpha_k, f_k(0))$. Given the uniqueness of the bivariate polynomial S, an equivalent way of defining the polynomial $q'(x)$ that $\mathcal{S}$ sends to the trusted party is via the points $(\alpha_k, S(0, \alpha_k))$. Since $\mathcal{S}$ sends $q'(x)$ to the trusted party in an ideal execution, we have that all honest parties P_j output $q'(\alpha_j)$ in an ideal execution. We now prove that the same also holds in a real protocol execution.

We stress that the polynomial $q'(x)$ is defined as a deterministic function of the transcript of messages sent by $\mathcal{A}$ in a real or ideal execution. Furthermore, since the execution is deterministic, the exact same polynomial $q'(x)$ is defined in both the real and ideal executions. It therefore remains to show that each honest party P_j outputs $q'(\alpha_j)$ in a real execution. We first observe that any honest party P_k for $k \in K$ clearly outputs $q'(\alpha_k)$. This follows from the fact that by the protocol specification, each party P_i that does not output $\bot$ outputs $f_i(0)$. Thus, each such P_k outputs $f_k(0)$. We have already seen that $q'(x)$ is the unique polynomial that passes through the points $(\alpha_k, f_k(0))$ and thus $q'(\alpha_k) = f_k(0)$ for every $k \in K$.

It remains to show that every honest party P_j for $j \notin K$ also outputs $q'(\alpha_j)$. Let $F_j(x)$ and $G_j(x)$ be the polynomials that P_j holds at the end of the computation (note that these polynomials may be different from the original polynomials that P_j received from the dealer at the first stage). We stress that this party P_j may or may not have broadcast good, and therefore we cannot rely on the conditions above. However, for every party P_k $(k \in K)$ who broadcast good, we are guaranteed that the polynomials $f_k(x)$ and $g_k(y)$ are consistent with the values of the polynomials of P_j. That is, it holds that $f_k(\alpha_j) = G_j(\alpha_k)$ and $g_k(\alpha_j) = F_j(\alpha_k)$. This follows from the fact that all conflicts are properly resolved (and so if they are inconsistent a reveal message must have been sent to make them consistent). This implies that for $t+1$ points $k \in K$, it holds that $F_j(\alpha_k) = S(\alpha_k, \alpha_j)$. Now, since $F_j(x)$ is a polynomial of degree t (by the truncation instruction; see the protocol specification), it follows that $F_j(x) = S(x, \alpha_j)$. Thus, $F_j(0) = S(0, \alpha_j)$ and we have that P_j outputs $S(0, \alpha_j)$. This completes the proof because $q'(\alpha_j) = S(0, \alpha_j)$, as described above. ■

This completes the proof of Theorem 5.5. ■

Efficiency. We remark that in the case that no parties behave maliciously in Protocol 5.4, the protocol merely involves the dealer sending two polynomials to each party, and then each party sending two field elements to every other party. We stress that when a secure broadcast protocol is used, all parties need to wait in round 3 the appropriate number of rounds in order to be sure that no party indeed broadcast a complaint; however, no messages need to be sent. Thus, using a broadcast that requires $t+1$ rounds, we have that in the *"good"* case where no one attempts to cheat, Protocol 5.4 runs in $t+3$ rounds, and each party just needs to send and receive $2n$ field elements (beyond the 2 polynomials sent by the dealer to every party).

6 Multiplication in the Presence of Malicious Adversaries

In this section, we show how to securely compute shares of the product of shared values, in the presence of a malicious adversary corrupting any $t < n/3$ parties. We use the simplification of the original multiplication protocol of [4] that appears in [16]; this simplification is described in detail in Section 6.4. As in the semi-honest case, the multiplication protocol works by having the parties compute a linear function of the product of their shares. The main problem and difficulty that arises in the case of malicious adversaries is that corrupted parties may not input the correct values. Since the local product of shares of degree-t polynomials define a polynomial of degree-$2t$, there is not enough redundancy to correct such errors.

The solution to this problem is as follows:

1. The parties first distribute shares of their inputs shares (on each wire) to all other parties, in a verifiable way. Observe that the input shares are from a degree-t polynomial. There is therefore enough redundancy to correct errors and so any incorrect values provided by corrupted parties are corrected. This operation is carried out using the $F_{VSS}^{subshare}$ functionality, described in Section 6.1.

2. Next, each party distributes shares of the product of its two input shares. Since all other parties already hold shares of the individual input shares, it is possible for them to verify that the shares received now are on a degree-t polynomial with the appropriate free coefficient. This operation is carried out using the F_{VSS}^{mult} functionality, described in Section 6.3. We remark that in order to compute F_{VSS}^{mult}, we need to use another functionality called F_{eval} that is first defined and constructed in Section 6.2.

3. Finally, after the previous step, all parties verifiably hold degree-t shares of the product of the input shares of every party. This enables them to multiply easily, using the method of [16]. This is the final multiplication protocol and it is described in Section 6.4.

6.1 The $F_{VSS}^{subshare}$ Functionality for Sharing Shares

Defining the functionality. We begin by defining the $F_{VSS}^{subshare}$ functionality. Informally speaking, this functionality is a way for a set of parties to verifiably give out shares of values that are themselves shares. Specifically, assume that the parties $P_1, \ldots, P_n$ hold values $f(\alpha_1), \ldots, f(\alpha_n)$, respectively, where f is a degree-t polynomial. The aim of the parties is for each to *share its share* $f(\alpha_i)$ with all other parties. In the semi-honest setting, this can be achieved simply by having each party P_i choose a random polynomial $g_i(x)$ with free coefficient $f(\alpha_i)$ and then send each P_j the share $g_i(\alpha_j)$. However, in the malicious setting a number of problems may arise. First, we must ensure that the corrupted parties choose polynomials $g_i(x)$ of degree-t; this can be solved using VSS. Second, we must ensure that a corrupted party P_i uses a polynomial $g_i(x)$ with the correct free coefficient $f(\alpha_i)$, where $f(x)$ is the degree-t polynomial defined by the honest parties' inputs (we stress that since there are more than t honest parties, the free coefficients of their input polynomials fully determine $f(x)$). This is much more problematic, and is the main challenge in implementing this functionality. We remark that in the case that a corrupted party P_i does not provide a valid input (i.e., it does not input a degree-t polynomial $g_i(x)$ such that $g_i(0) = f(\alpha_i)$), the functionality defines a *new polynomial* $g'_i(x)$ which is the constant polynomial $g'_i(x) = f(\alpha_i)$ for all x, and uses this in place of $g_i(x)$ in the outputs. In addition, the corrupted parties may choose their input polynomials after seeing shares of $g_j(x)$ for $j \notin I$. Nevertheless, this suffices for its use in the BGW protocol. Therefore, we explicitly define the functionality so that this is allowed. This requires a definition of a *reactive functionality*; see Section 2.2.

FUNCTIONALITY 6.1 (The reactive $F_{VSS}^{subshare}$ functionality)

1. The $F_{VSS}^{subshare}$ functionality receives the inputs of the honest parties $\{\beta_j\}_{j\notin I}$. Let $f(x)$ be the unique degree-t polynomial determined by the points $\{(\alpha_j, \beta_j)\}_{j\notin I}$.[6]
2. For every $j \notin I$, $F_{VSS}^{subshare}$ chooses a random degree-t polynomial $g_j(x)$ under the constraint that $g_j(0) = \beta_j = f(\alpha_j)$.

   $F_{VSS}^{subshare}$ sends shares $\{g_j(\alpha_i)\}_{j\notin I; i\in I}$ to the (ideal) adversary.
3. $F_{VSS}^{subshare}$ receives polynomials from the corrupted parties $\{g_i(x)\}_{i\in I}$; if a polynomial $g_i(x)$ is not received, then $F_{VSS}^{subshare}$ sets $g_i(x) = 0$.
4. $F_{VSS}^{subshare}$ *determines the output polynomials* $g'_1(x), \ldots, g'_n(x)$*:*
   (a) For every $j \notin I$, $F_{VSS}^{subshare}$ sets $g'_j(x) = g_j(x)$.
   (b) For every $i \in I$, $F_{VSS}^{subshare}$ checks that $g_i(0) = f(\alpha_i)$ and that $\deg(g_i) = t$. If yes, it sets $g'_i(x) = g_i(x)$. Else, it sets $g'_i(x) = f(\alpha_i)$; i.e., $g'_i(x)$ is the constant polynomial equalling $f(\alpha_i)$ everywhere.
5. $F_{VSS}^{subshare}$ sends the polynomial $g'_k(x)$ and the shares $(g'_1(\alpha_k), \ldots, g'_n(\alpha_k))$ to party P_k, for every $k = 1, \ldots, n$.

Background to implementing $F_{VSS}^{subshare}$. Let $G \in \mathbb{F}^{(t+1)\times n}$ be the generator matrix for a (generalized) Reed-Solomon code of length $n = 3t+1$, dimension $k = t+1$ and distance $d = 2t+1$. In matrix notation, the encoding of a vector $\vec{a} = (a_0, \ldots, a_t) \in \mathbb{F}^{t+1}$ is given by $\vec{a} \cdot G$, where:

$$G \stackrel{\text{def}}{=} \begin{pmatrix} 1 & 1 & \ldots & 1 \\ \alpha_1 & \alpha_2 & \ldots & \alpha_n \\ \vdots & \vdots & & \vdots \\ \alpha_1^t & \alpha_2^t & \ldots & \alpha_n^t \end{pmatrix}.$$

Letting $f(x) = \sum_{k=0}^{t} a_0 \cdot x^t$ be a degree-t polynomial, we have that the Reed-Solomon encoding of $\vec{a}$ is the vector $\langle f(\alpha_1), \ldots, f(\alpha_n)\rangle$. Let $H \in \mathbb{F}^{2t\times n}$ be the parity-check matrix of G; the matrix is of

[6]If not all the points lie on a single degree-t polynomial, then no security guarantees are obtained. Formally, this is achieved by defining that in this case the functionality sends the inputs of the honest parties to the corrupted parties, and sets the output of the honest parties to be whatever the adversary desires. In this way, any protocol is secure in this "bad case". From now on we just ignore this case, since in all uses of the functionality we have that all of the honest parties' points lie on a single degree-t polynomial.

the form:

$$H = \begin{pmatrix} 1 & 1 & \dots & 1 \\ \alpha_1 & \alpha_2 & \dots & \alpha_n \\ \vdots & \vdots & & \vdots \\ \alpha_1^{2t-1} & \alpha_2^{2t-1} & \dots & \alpha_n^{2t-1} \end{pmatrix} \cdot \begin{pmatrix} v_1 & & 0 \\ & \ddots & \\ 0 & & v_n \end{pmatrix} \tag{10}$$

for non-zero values $v_1, \dots, v_n$ such that $G \cdot H^T = 0$. The syndrome of a word $\vec{y} \in \mathbb{F}^n$ is given by $S(\vec{y}) = \vec{y} \cdot H^T \in \mathbb{F}^{2t}$. A basic fact from error-correcting codes is that for any $y = \vec{a} \cdot G$, $S(\vec{y}) = 0^{2t}$.

The protocol. In the protocol, each party P_i is instructed to choose a random polynomial $g_i(x)$ under the constraint that $g_i(0) = f(\alpha_i)$, as in the simple semi-honest protocol described above. The parties then share the $g_i(x)$ polynomials using F_{VSS}; unlike the semi-honest case, corrupted parties may share polynomials $g_i(x)$ with arbitrary free coefficients. Now, let $\vec{y} \in \mathbb{F}^n$ be the vector $(y_1, \dots, y_n) = (g_1(0), \dots, g_n(0))$. For all honest parties P_j it is guaranteed that $g_j(0) = f(\alpha_j)$, whereas there is no guarantee regarding the values $g_i(0)$ for corrupted P_i. It follows that $\vec{y}$ is a word that is at most distance t from the vector $\langle f(\alpha_1), \dots, f(\alpha_n)\rangle$ which is a Reed-Solomon codeword of length $n = 3t+1$. Thus, in principle, it is possible to correct the word $\vec{y}$ using Reed-Solomon error correction. Since the protocol cannot reveal the actual values $f(\alpha_j)$, this error correction is carried out from the shares themselves. Specifically, in the protocol, the parties compute the syndrome $S(\vec{y})$ from their shares. The crucial observation is that since the computation of the syndrome is just matrix multiplication, it is a linear function. Therefore, *shares of the syndrome* can be computed locally from *shares of the points* $y_1, \dots, y_n$. Furthermore, as is shown in the proof, the syndrome itself can be computed by adversary in the ideal model. Thus, the parties can all send their shares of the syndrome to all other parties, enabling each to reconstruct the syndrome $S(\vec{y})$. This then enables the parties to run the Reed-Solomon error-correcting procedure and obtain the error vector (the error vector $\vec{e} = (e_1, \dots, e_n)$ is such that for every i, $y_i - e_i = f(\alpha_i)$); we remark that the error vector can be computed from the syndrome alone. This error vector then provides the honest parties all the information that they need to compute the output. Specifically, if $e_i = 0$, then this implies that P_i used a "correct" polynomial $g_i(x)$ for which $g_i(0) = f(\alpha_i)$, and so the parties can just output the shares $g_i(\alpha_j)$ that they received in the F_{VSS} sharing at the onset. In contrast, if $e_i \neq 0$ then the parties know that P_i is malicious, and can all send each other the shares $g_i(\alpha_j)$ that they received in the F_{VSS} sharing. This enables them to reconstruct the polynomial $g_i(x)$ and compute $g_i(0) - e_i = f(\alpha_i)$. Thus, they obtain the actual share of the corrupted party, as required in the functionality definition. See Protocol 6.3 for the full specification.

Theorem 6.2 *Let $t < n/3$. Then, Protocol 6.3 t-securely computes the $F_{VSS}^{subshare}$ functionality in the F_{VSS}-hybrid model, in the presence of a static malicious adversary.*

Proof: Intuitively, the simulator $\mathcal{S}$ that we construct receives the outputs $\{g'_j(\alpha_i)\}_{j \notin I}$ for every $i \in I$ and sends these to the real adversary $\mathcal{A}$ as its output from each F_{VSS} invocation with the honest P_j as dealer. It then receives the polynomials $\{g_i(x)\}_{i \in I}$ as the corrupted parties' inputs to the F_{VSS} invocations in which a corrupted P_i plays dealer; $\mathcal{S}$ sends these polynomials to the trusted party computing $F_{VSS}^{subshare}$. Finally, $\mathcal{S}$ receives back the outputs of the corrupted parties, and uses these outputs to generate the messages that the real adversary $\mathcal{A}$ would receive from the honest parties in a real execution of Protocol 6.3. The main challenge is to show how $\mathcal{S}$ can compute the syndrome values correctly from the corrupted parties' input/output values.

PROTOCOL 6.3 (Securely computing $F_{VSS}^{subshare}$ in the F_{VSS}-hybrid model)

- **Inputs:** Each party P_i holds a value β_i; we assume that the points (α_j, β_j) for every honest P_j all lie on a single degree-t polynomial (see the definition of $F_{VSS}^{subshare}$ above and Footnote 6).
- **Common input:** The description of a field $\mathbb{F}$ and n specified elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$.
- **The protocol:**
  1. Each party P_i chooses a random degree-t polynomial $g_i(x)$ under the constraint that $g_i(0) = \beta_i$, and invokes the F_{VSS} functionality as dealer with $g_i(x)$ as its private input.
  2. At the end of this stage, each party P_i holds the values $g_1(\alpha_i), \ldots, g_n(\alpha_i)$. If any value is missing (or equals $\perp$), P_i replaces it with 0.
  3. For every $r = 1, \ldots, 2t$, each party P_i computes:
  $$S_r(\alpha_i) = \sum_{k=1}^{n} g_k(\alpha_i) \cdot v_k \cdot (\alpha_k)^{r-1}$$
  Observe that $S_r(0) = s_r$, the rth value in the syndrome of the word $(g_1(0), \ldots, g_n(0))$.
  4. Each party P_i sends $S_r(\alpha_i)$ for every $(1 \leq r \leq 2t)$ to every P_j $(1 \leq j \leq n)$.
  5. At this stage, each party P_k holds $S_r(\alpha_1), \ldots, S_r(\alpha_n)$ for every $r = 1, \ldots, 2t$ (if any value is missing, it replaces it with 0). P_i uses the Reed-Solomon decoding procedure (with $d = 2t + 1$) to reconstruct the polynomial $S_r(x)$ for every $r = 1, \ldots, 2t$.
  6. Let $s_r = S_r(0)$ and define the syndrome to be $\vec{s} = (s_1, \ldots, s_{2t})$. Each party locally runs the Reed-Solomon decoding procedure using $\vec{s}$ only, and receives back an error vector $\vec{e} = (e_1, \ldots, e_n)$.
  7. For every k such that $e_k = 0$: each party P_i sets $g'_k(\alpha_i) = g_k(\alpha_i)$.
  8. For every k such that $e_k \neq 0$:
     (a) Each party P_i sends $g_k(\alpha_i)$ to every P_j.
     (b) Each party P_i receives $g_k(\alpha_1), \ldots, g_k(\alpha_n)$; if any value is missing, it sets it to 0. P_i runs the Reed-Solomon decoding procedure on these values to reconstruct $g_k(x)$.
     (c) Each party P_i computes $g_k(0)$, and sets $g'_k(\alpha_i) = g_k(0) - e_k$ (which equals $f(\alpha_k)$).
  9. P_i outputs $g_i(x)$ and $g'_1(\alpha_i), \ldots, g'_n(\alpha_i)$.

We now proceed to provide a full description of the simulator.

The simulator $\mathcal{S}$:

1. *$\mathcal{S}$ invokes $\mathcal{A}$ with its auxiliary input z.*
2. *$\mathcal{S}$ receives the values $\{g_j(\alpha_i)\}_{j \notin I; i \in I}$ from the trusted party computing $F_{VSS}^{subshare}$.*
3. *For every $i \in I$ and $j \notin I$, the simulator $\mathcal{S}$ simulates the honest party P_j sending $g_j(\alpha_i)$ to P_i (where $g_j(\alpha_i)$ is the appropriate value received from $F_{VSS}^{subshare}$ in the previous step).*

4. *For every $i \in I$, the simulator $\mathcal{S}$ receives from $\mathcal{A}$ the polynomial $g_i(x)$ that $\mathcal{A}$ sends as input to the F_{VSS} functionality with P_i as dealer. If $\deg(g_i(x)) > t$ or was not sent by $\mathcal{A}$, then $\mathcal{S}$ replaces it with the constant polynomial $g_i(x) = 0$.*

5. *For every $i \in I$, $\mathcal{S}$ instructs party P_i in the ideal model to send $g_i(x)$ to $F_{VSS}^{subshare}$.*

6. *$\mathcal{S}$ receives back outputs $g'_i(x)$ and $(g'_1(\alpha_i), \ldots, g'_n(\alpha_i))$ for every $i \in I$.*

7. $\mathcal{S}$ generates the messages of the honest parties: *$\mathcal{S}$ uses the outputs that it received in order to generate the view that $\mathcal{A}$ would have in a real execution. It does this, according to the following steps:*

   (a) Construct the error vector $\vec{e}$ for decoding:
   - *For every $j \notin I$, $\mathcal{S}$ sets $e[j] = 0$.*
   - *For every $i \in I$:*
     - *If $g'_i(x) = g_i(x)$, then $\mathcal{S}$ sets $e[i] = 0$ (recall that $g'_i(x)$ is part of the output received by $\mathcal{S}$ and $g_i(x)$ is the input used by $\mathcal{A}$ to F_{VSS})*
     - *If $g'_i(x) \neq g_i(x)$, then $\mathcal{S}$ sets $e[i] = g'_i(0) - g_i(0)$ (note that $e[i] = f(\alpha_i) - g_i(0)$ because, by the definition of $F_{VSS}^{subshare}$, in this case $g'_i(x) = f(\alpha_i)$ is a constant polynomial)*

   (b) Compute the syndrome: *$\mathcal{S}$ computes $(s_1, \ldots, s_{2t}) = S(\vec{e}) = \vec{e} \cdot H^T$, where H is the parity-check matrix of the code, as described above.*

   (c) Find the polynomials $S_r(x)$: *For every $r = 1, \ldots, 2t$, $\mathcal{S}$ chooses a random polynomial $S_r(x)$ of degree t such that:*

      i. *$S_r(0) = s_r$; the rth value of the syndrome*

      ii. *For every $i \in I$, $S_r(\alpha_i) = \sum_{k=1}^{n} g_k(\alpha_i) \cdot v_k \cdot (\alpha_k)^{r-1}$*

   Observe that if $|I| = t$ then the constraints define $t+1$ points and so the polynomial $S_r(x)$ is fully determined.

8. Send the messages that honest parties would send:

   (a) *For every $j \notin I$, $\mathcal{S}$ simulates P_j sending $S_1(\alpha_j), \ldots, S_{2t}(\alpha_j)$ to all parties.*

   (b) *For every $i \in I$ for which $e[i] \neq 0$ and every $j \notin I$, $\mathcal{S}$ simulates P_j sending $g_i(\alpha_j)$ to all parties.*

We remark that $\mathcal{S}$ must compute the actual syndrome (and not just shares of it), because the parties all receive the reconstructed polynomials $S_r(x)$ in the protocol, for $r = 1, \ldots, 2t$. The idea behind the way that $\mathcal{S}$ constructs the syndrome is based on the fact that the syndrome of any word equals the syndrome of the error vector. Since $\mathcal{S}$ can compute the error vector exactly (it knows that there is no error for every $j \notin I$ and it can check if $g'_i(x) = g_i(x)$ for every $i \in I$), $\mathcal{S}$ can compute the syndrome of the word correctly. This enables it to then generate the appropriate distribution, as described. We now prove that for every $I \subset [n]$ with $|I| \leq t$:

$$\left\{\text{IDEAL}_{F_{VSS}^{subshare}, \mathcal{S}(z), I}(\vec{x})\right\}_{z \in \{0,1\}^*; \vec{x} \in \mathbb{F}^n} \equiv \left\{\text{HYBRID}_{\pi, \mathcal{A}(z), I}^{F_{VSS}}(\vec{x})\right\}_{z \in \{0,1\}^*; \vec{x} \in \mathbb{F}^n}.$$

We first show that the outputs of the honest parties are distributed identically in the real and ideal executions. Then, we show that the view of the corrupted parties, given the output of the honest parties, is distributed identically in the real and ideal executions. This implies that the joint distribution of the outputs of all parties in the real and ideal executions are identical.

The honest parties' outputs. In a real protocol execution, each party P_j distributes shares of its chosen polynomial $g_j(x)$, using the F_{VSS} functionality. Then, each party P_j locally computes the values $S_1(\alpha_j), \ldots, S_{2t}(\alpha_j)$, where for each $r = 1, \ldots, 2t$ it holds that:

$$S_r(x) = \sum_{k=1}^{n} g_k(x) \cdot v_k \cdot (\alpha_k)^{r-1}. \tag{11}$$

Note that although each honest party P_j computes $S_r(\alpha_j)$ locally, they all compute relative to the same polynomial $S_r(x)$. This is guaranteed since the $g_k(x)$ polynomials are shared using F_{VSS}. Thus, there exist degree-t polynomials $g_1(x), \ldots, g_n(x)$ such that *every* honest party P_j receives $\{g_k(\alpha_j)\}_{k\in[n]}$ (or $\perp$ in which case all honest parties uses the share 0).

After each P_j locally computes the $S_r(\alpha_j)$ values, the parties all send these values to each other, and each party locally reconstructs the polynomials $S_1(x), \ldots, S_{2t}(x)$. Observe that since all the polynomials $g_i(x)$ are of degree t, the shares $S_r(\alpha_i)$ are from a polynomial of degree t. Thus, for every $r = 1, \ldots, 2t$, each honest party holds a vector $(\sigma_r^1, \ldots, \sigma_r^n)$ where for every $j \notin I$ it is guaranteed that $\sigma_r^j = S_r(\alpha_j)$. Stated differently, each honest party holds a word that is at most distance t from a Reed-Solomon codeword $\langle S_r(\alpha_1), \ldots, S_r(\alpha_n)\rangle$ of length $n = 3t + 1$. Thus, using the Reed-Solomon decoding procedure, each honest party obtains the same polynomial $S_r(x)$, as defined in Eq. (11).

Next, each honest party computes $s_r = S_r(0)$ and constructs the vector $(s_1, \ldots, s_{2t})$. Let $\vec{y} = (g_1(0), \ldots, g_n(0))$. We now claim that the syndrome $S(\vec{y}) = \vec{y} \cdot H^T$ of $\vec{y}$ is exactly the vector $(s_1, \ldots, s_{2t})$. In order to see this observe that by Eq. (10) the rth element of $S(\vec{y})$ equals $\sum_{k=1}^{n} g_k(0) \cdot v_k \cdot (\alpha_k)^{r-1}$, which is exactly $S_r(0)$ as defined in Eq. (11). Under the assumption (see Footnote 6) that there exists a degree-t polynomial $f(x)$ such that for every $j \notin I$ it holds that $g_j(0) = f(\alpha_j)$, we have that $\vec{y}$ is at most distance t from the Reed-Solomon codeword $\langle f(\alpha_1), \ldots, f(\alpha_n)\rangle$. Thus, it is possible to efficiently compute the error vector $(e_1, \ldots, e_n)$ from the syndrome $(s_1, \ldots, s_{2t})$. By definition, this error vector has the property that for every k, $y_k - e_k = f(\alpha_k)$. Equivalently, we have that for every $k \in [n]$,

$$g_k(0) - e_k = f(\alpha_k).$$

We are now ready to analyze the output distribution of the honest parties in the real and ideal executions. Clearly, the polynomial $g'_j(x)$ output by every honest party P_j is distributed identically in the real and ideal executions, because in each case $g_j(x)$ is a random polynomial with free coefficient $\beta_j = f(\alpha_j)$ and it always holds that $g'_j(x) = g_j(x)$ for an honest P_j. We now proceed to analyze the distribution over the shares $(g'_1(\alpha_j), \ldots, g'_n(\alpha_j))$ received by every honest P_j. We separately consider the shares $g'_k(\alpha_j)$ where P_k is honest and where P_k is corrupted:

For every honest party P_k ($k \notin I$) we have that by our assumption $g_k(0) = f(\alpha_k)$. Therefore, $y_k = g_k(0)$ is a correct element in the codeword and $e_k = 0$. Thus, every honest party P_j sets $g'_k(\alpha_j) = g_k(\alpha_j)$. Furthermore, by the protocol specification, honest party P_k chose $g_k(0)$ to be a *random polynomial* with free coefficient $f(\alpha_k)$. Thus, the polynomial $g'_j(x)$ and values $\{g'_k(\alpha_j)\}_{k\notin I}$ output by an honest P_j are identically distributed to these analogous values in the output generated by the trusted party computing $F_{VSS}^{subshare}$ (since $F_{VSS}^{subshare}$ chooses the $g'_k(x)$ polynomials for honest P_k in the same way).

We now proceed to consider the values $g'_k(\alpha_j)$ for corrupted P_k ($k \in I$). There are two cases, depending on the value of e_k. We stress that since the Reed-Solomon decoding can correct up to t incorrect values, each honest party obtains the same polynomials $S_1(x), \ldots, S_{2t}(x)$. Thus, all the honest parties obtain the same value of e_k. We have the following cases:

1. *Case 1 – $e_k = 0$:* By the property of the error vector, this implies that $g_k(0) = f(\alpha_k)$. Thus, in the ideal model, the trusted party computing $F_{VSS}^{subshare}$ sets $g'_k(x) = g_k(x)$ and each honest P_j receives the share $g'_k(\alpha_j)$. (This holds as long as $g_k(x)$ is of degree t. If $g_k(x)$ is not of degree t, then the output share of each P_j determined by $F_{VSS}^{subshare}$ for every P_j is the constant value $f(\alpha_k)$. In contrast, in the real execution, if $g_k(x)$ is of degree greater than t, the output from the F_{VSS} execution in the protocol is $\perp$ to each honest party. Thus, the honest parties all set their values to 0 and so $g_k(x)$ is taken to be the constant polynomial $g_k(x) = 0$. Since we are in the case that $e_k = 0$ this implies that $f(\alpha_k) = 0$. Thus, in the real model each honest party P_j outputs $g'_k(\alpha_j) = 0$, which is the same as the ideal model because $f(\alpha_k) = 0$.)

2. *Case 2 – $e_k \neq 0$:* In this case, each honest party P_j sends $g_k(\alpha_j)$ to all others. The parties then reconstruct the polynomial $g_k(x)$. As before, since $g_k(x)$ is distributed using F_{VSS}, we are guaranteed that the honest parties hold correct shares and therefore there are at least $2t+1$ correct shares of the polynomial $g_k(x)$. Thus, the Reed-Solomon error correction procedure will result in the exact polynomial $g_k(x)$. All honest parties then output the same fixed value $g_k(0) - e_k$. Since $g_k(0) - e_k = f(\alpha_k)$, it follows that all honest parties output the same value $f(\alpha_k)$. Now, since $e_k \neq 0$ it follows that $g_k(0) \neq f(\alpha_k)$ and so in the ideal model, $F_{VSS}^{subshare}$ sets $g'_k(x) = f(\alpha_k)$. This implies that, exactly as in a real execution, all honest parties output the same value α_k.

The adversary's view. We now show that the view of the adversary $\mathcal{A}$ is identically distributed in the real and ideal executions, given the outputs of the honest parties. Fix $\{g'_j(x)\}_{j \notin I}$ and $\{g'_k(\alpha_j)\}_{j \notin I; k \in [n]}$ as the output of the honest parties in the real and ideal executions; i.e., honest party P_j receives $g'_j(x)$ and $(g'_1(\alpha_j), \ldots, g'_n(\alpha_j))$. Note that the output of the honest parties determines all of the polynomials, including $g'_i(x)$ for $i \in I$; this is due to the fact that the honest parties' outputs include more than t shares of each of these polynomials.

In the first step of the protocol, the parties choose random degree-t polynomials $g_j(x)$ such that $g_j(0) = \beta_j$. As we have shown above, in a real protocol execution it holds that $g'_j(x) = g_j(x)$ for every honest party P_j. Thus, the shares $\{g_j(\alpha_i)\}_{i \in I; j \notin I}$ received by the corrupted parties as output from the F_{VSS} sharing step in the protocol are computed from the polynomials $\{g'_j(x)\}_{j \notin I}$ which are fixed here from the given honest parties' output. Likewise, in the ideal simulation by $\mathcal{S}$, the shares received by $\mathcal{A}$ as output from the F_{VSS} step are exactly those that $\mathcal{S}$ received from $F_{VSS}^{subshare}$ as output, which are computed from the polynomials $\{g_j(x)\}_{j \notin I}$. Thus, when conditioning on the output of the honest parties, the shares received by the corrupted parties in a real protocol execution and in the simulation by $\mathcal{S}$ are exactly the same. (We stress that there is no randomness here; the values are exactly the same and not just distributed identically.)

In the next step of the protocol, the parties all send shares of the polynomials $S_1(x), \ldots, S_{2t}(x)$. In the simulation by $\mathcal{S}$, the polynomials $S_r(x)$ are computed differently and we therefore have to show that they have the same distribution. In the protocol, the parties compute and send the shares $S_r(\alpha_i) = \sum_{k=1}^{n} g_k(\alpha_i) \cdot v_k \cdot (\alpha_k)^{r-1}$, and then reconstruct the polynomials:

$$S_r(x) = \sum_{k=1}^{n} g_k(x) \cdot v_k \cdot (\alpha_k)^{r-1}.$$

As we have seen, for every $r = 1, \ldots, 2t$ it holds that $S_r(0) = \vec{y} \cdot H$. Furthermore, by the property of the error vector in error-correcting codes it holds that $\vec{e} \cdot H = \vec{y} \cdot H$ and so $S_r(0) = \vec{e} \cdot H$. Thus, the error vector $\vec{e}$ computed by $\mathcal{S}$ (by setting $e[i] = f(\alpha_i) - g_i(0)$ for every $i \in I$) yields exactly the same syndrome string $(s_1, \ldots, s_{2t})$ as that computed by the honest parties in a real execution. Note that $\mathcal{S}$ generates $(s_1, \ldots, s_{2t})$ from $\vec{e}$ whereas the honest parties compute $\vec{e}$ from $(s_1, \ldots, s_{2t})$. There are two cases:

1. *Case 1 – $|I| = t$:* In this case, the syndrome $(s_1, \ldots, s_{2t})$ and the values $\{S_r(\alpha_i)\}_{i \in I; r \in [2t]}$, which can all be computed by $\mathcal{S}$, fully determine the polynomials $S_1(x), \ldots, S_{2t}(x)$. Furthermore, these values are all fully determined from the polynomials $\{g_i(x)\}_{i \in I}$ sent by $\mathcal{A}$ in the F_{VSS} step and by the polynomials $\{g'_j(x)\}_{j \notin I}$ in the output of the honest parties. Since the values received by $\mathcal{A}$ in the first step of the simulation are identical in the real and ideal executions (conditioned on the honest parties' outputs), it follows that the polynomials $\{g_i(x)\}_{i \in I}$ and $\{g'_j(x)\}_{j \notin I}$ are identical, and so the polynomials $S_1(x), \ldots, S_{2t}(x)$ are also identical in the real and ideal executions. We therefore conclude that the view of $\mathcal{A}$ in a real execution is identical to its view in an ideal execution (where both are conditioned on the outputs of the honest parties). We stress that in this case, the views are identical in the sense that the string generated by $\mathcal{S}$ contains the exact same messages that $\mathcal{A}$ would receive in a real execution. This is because when $|I| = t$ and the honest parties' outputs are fixed, there is no randomness at all.

2. *Case 2 – $|I| < t$:* In this case, for every $r = 1, \ldots, 2t$, $\mathcal{S}$ chooses a random polynomial $S_r(x)$ of degree-t such that $S_r(0) = s_r$ and for every $i \in I$, $S_r(\alpha_i) = \sum_{k=1}^{n} g_k(\alpha_i) \cdot vk \cdot (\alpha_k)^{r-1}$. In contrast, in a real execution, $S_r(x)$ is computed also from points $S_r(\alpha_j)$ for some $j \notin I$. Specifically, in order to fully determine $S_r(x)$, it is necessary to add $t - |I|$ points $(\alpha_j, S_r(\alpha_j))$. Furthermore, $S_r(\alpha_j)$ is a function of $g_k(\alpha_j)$ for every $k \in [n]$ and in particular for the unknown $g_k(\alpha_j)$ with $k \notin I$. Thus, $S_r(x)$ is determined by $t - |I|$ additional points $(\alpha_j, g_k(\alpha_j))$ for every $k \notin I$ (i.e., for each of these k, we need $(\alpha_j, g_k(\alpha_j))$ for $t - |I|$ different values of $j \notin I$). However, by Corollary 3.6, we have that for $k \notin I$ the values $g_k(\alpha_j)$ are uniformly distributed.[7] This implies that $S_r(\alpha_j)$ is uniformly distributed, and so the distribution over the polynomials $S_r(x)$ chosen by $\mathcal{S}$ in the simulation is identical to those computed in a real protocol execution.

The proof is concluded by noting that as long as the $S_r(x)$ polynomials are correctly distributed and the error vector $\vec{e}$ is correct, the messages sent by $\mathcal{S}$ are exactly the same as those sent by the honest parties in a real protocol execution. ■

[7]Actually, Corollary 3.6 refers to the case that none of the points are given, and here $|I|$ out of the t are given. Nevertheless, it trivially follows when conditioning on some of the points because the random variables in the corollary are independent.

6.2 The F_{eval} Functionality for Evaluating a Shared Polynomial

In the protocol for verifying the multiplication of shares, the parties need to process complaints by evaluating shared polynomials at the point of the complaining party. Specifically, given shares $f(\alpha_1), \ldots, f(\alpha_n)$, of a polynomial f, the parties need to compute $f(\alpha_k)$ for a predetermined k, without revealing anything else. We begin by formally defining the functionality; the functionality is parameterized by an index k that determines at which point the polynomial is to be evaluated.

FUNCTIONALITY 6.4 (The F^k_{eval} functionality)

1. The F^k_{eval} functionality receives the inputs of the honest parties $\{\beta_j\}_{j \notin I}$. Let $f(x)$ be the unique degree-t polynomial determined by the points $\{(\alpha_j, \beta_j)\}_{j \notin I}$. (If not all the points lie on a single degree-t polynomial, then no security guarantees are obtained; see Footnote 6.)
2. The functionality F^k_{eval} sends the output pair $(f(\alpha_i), f(\alpha_k))$ to every party P_i, for $i = 1, \ldots, n$.

Equivalently, in function notation, we have:

$$F^k_{eval}\Big(\beta_1, \ldots, \beta_n\Big) = \Big(((f(\alpha_1), f(\alpha_k)), \ldots, (f(\alpha_n), f(\alpha_k))\Big)$$

where f is the result of Reed-Solomon decoding on $(\beta_1, \ldots, \beta_n)$. We remark that although each party P_i already holds $f(\alpha_i)$ as part of its input, we need the output to include this value in order to simulate in the case that a corrupted party has incorrect input. This will not make a difference in its use, since $f(\alpha_i)$ is supposed to be known to P_i in any case.

Background. The parties' inputs are a vector $\vec{\beta} \stackrel{\text{def}}{=} (\beta_1, \ldots, \beta_n)$ where for every $j \notin I$ it holds that $\beta_j = f(\alpha_j)$. Thus, the parties' inputs are computed by

$$\vec{\beta} = V_{\vec{\alpha}} \cdot \vec{f},$$

where $V_{\vec{\alpha}}$ is the Vandermonde matrix, and $\vec{f}$ is the vector of coefficients for the polynomial $f(x)$. We remark that $\vec{f}$ is of length n, and is padded with zeroes beyond the $(t+1)$th entry. Let $\vec{\alpha}_k = (1, \alpha_k, (\alpha_k)^2, \ldots, (\alpha_k)^{n-1})$ be the kth row of $V_{\vec{\alpha}}$. Then the output of the functionality is

$$f(\alpha_k) = \vec{\alpha}_k \cdot \vec{f}.$$

We have:

$$\vec{\alpha}_k \cdot \vec{f} = \vec{\alpha}_k \cdot \left(V_{\vec{\alpha}}^{-1} \cdot V_{\vec{\alpha}}\right) \cdot \vec{f} = \left(\vec{\alpha}_k \cdot V_{\vec{\alpha}}^{-1}\right) \cdot \Big(V_{\vec{\alpha}} \cdot \vec{f}\Big) = \left(\vec{\alpha}_k \cdot V_{\vec{\alpha}}^{-1}\right) \cdot \vec{\beta}$$

and so there exists a vector of *constants* $(\vec{\alpha}_k \cdot V_{\vec{\alpha}}^{-1})$ so that the inner product of this vector and the inputs yields the desired result. In other words, F^k_{eval} is simply a linear function of the parties' inputs.

The protocol. Since F^k_{eval} is simply a linear function of the parties' inputs, it can be computed by each party sharing its share and then locally computing the function on the shares. The result is that each party P_i holds a share δ_i of a polynomial whose free coefficient is the result $f(\alpha_k)$. Thus, the parties can now simply send their δ_i shares and reconstruct the resulting polynomial.

In order to prevent malicious parties from cheating, the $F_{VSS}^{subshare}$ functionality is used in order to share the shares. Then, the reconstruction in the last stage is carried out using Reed-Solomon decoding; this ensures that $t < n/3$ malicious parties cannot affect the result. See Protocol 6.5 for the full description.

PROTOCOL 6.5 (Securely computing F_{eval}^{k} in the $F_{VSS}^{subshare}$-hybrid model)

- **Inputs:** Each party P_i holds a value β_i; we assume that the points (α_j, β_j) for every honest P_j all lie on a single degree-t polynomial f (see the definition of F_{eval}^{k} above and Footnote 6).
- **Common input:** The description of a field $\mathbb{F}$ and n specified elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$.
- **The protocol:**
  1. The parties invoke the $F_{VSS}^{subshare}$ functionality with each party P_i using β_i as its private input.
  2. At the end of this stage, each party P_i holds $g_1(\alpha_i), \ldots, g_n(\alpha_i)$, where all the $g_i(x)$ are of degree t, and for every i, $g_i(0) = f(\alpha_i)$.
  3. Each party P_i locally computes: $H(\alpha_i) = \sum_{\ell=1}^{n} \gamma_\ell \cdot g_\ell(\alpha_i)$, where $(\gamma_1, \ldots, \gamma_n) = \vec{\alpha}_k \cdot V_{\vec{\alpha}}^{-1}$. Each party P_i sends $H(\alpha_i)$ to all P_j.
  4. Upon receiving $(\hat{H}(\alpha_1), \ldots, \hat{H}(\alpha_n))$, each party runs the Reed-Solomon decoding procedure and receives $(H(\alpha_1), \ldots, H(\alpha_n))$. It then reconstructs $H(x)$ and computes $H(0)$.
  5. Each party P_i outputs $(\beta_i, H(0))$.

We have already provided the motivation behind the security of the protocol; we therefore proceed directly to the proof of security.

Theorem 6.6 *Let $t < n/3$. Then, Protocol 6.5 t-securely computes the F_{eval}^{k} functionality in the $F_{VSS}^{subshare}$-hybrid model, in the presence of a static malicious adversary.*

Proof: Intuitively, the simulator just sends random shares for the outputs that the corrupted parties expect to receive from the $F_{VSS}^{subshare}$ executions when the dealer is honest (when the dealer is dishonest, the simulator just answers exactly as $F_{VSS}^{subshare}$ would). Then, the simulator simulates the sending of shares of a random polynomial $H'(x)$ with free coefficient $f(\alpha_k)$, which it received as output from F_{eval}^{k}, and which agrees with the adversary's view (i.e., equals $\sum_{\ell=1}^{n} \gamma_\ell \cdot g_\ell(\alpha_i)$ at α_i, for every $i \in I$). The formal description follows:

The simulator $\mathcal{S}$:

1. *$\mathcal{S}$ invokes $\mathcal{A}$ with the auxiliary input z.*
2. *$\mathcal{S}$ receives from the trusted party the output values $\{(f(\alpha_i), f(\alpha_k))\}_{i \in I}$; observe that the corrupted parties provide no input to F_{eval}^{k}.*
3. *$\mathcal{S}$ simulates the $F_{VSS}^{subshare}$ invocations:*
   (a) *For every $j \notin I$, $\mathcal{S}$ chooses uniformly at random a polynomial $q'_j(x)$ from $\mathcal{P}^{0,t}$, and sends $\mathcal{A}$ the values $\left\{q'_j(\alpha_i)\right\}_{i \in I, j \notin I}$ as the corrupted parties' outputs of the first phase in the $F_{VSS}^{subshare}$; recall that $F_{VSS}^{subshare}$ is a reactive functionality.*

(b) *$\mathcal{S}$ receives from $\mathcal{A}$ the inputs $\{q_i(x)\}_{i\in I}$ of the corrupted parties to $F_{VSS}^{subshare}$. If $\mathcal{A}$ did not reply with some polynomial $q_i(x)$, then $\mathcal{S}$ sets $q'_i(x) = 0$.*

(c) *For every $i \in I$, $\mathcal{S}$ checks that $\deg(q_i) = t$ and $q_i(0) = f(\alpha_i)$. If this check fails, $\mathcal{S}$ sets $q'_i(x) = f(\alpha_i)$, where $f(\alpha_i)$ is part of the output received by $\mathcal{S}$ from F_{eval}^k (i.e, this simulates the case where $F_{VSS}^{subshare}$ functionality rejects the polynomial $q_i(x)$ and sets the output to be the constant polynomial equalling $f(\alpha_i)$ everywhere).*

(d) *$\mathcal{S}$ simulates each corrupted party P_i receiving output $q'_i(x)$ and $\{q'_\ell(\alpha_i)\}_{\ell\in I}$ from $F_{VSS}^{subshare}$.*

4. *$\mathcal{S}$ simulates the sending of the shares $H(\alpha_j)$:*

   (a) *$\mathcal{S}$ selects a random polynomial $H'(x)$ of degree t under the constraint that:*
      - *The free coefficient of $H'(x)$ is $f(\alpha_k)$; i.e., $H'(0) = f(\alpha_k)$.*
      - *$H'(x)$ passes through the points that the corrupted parties hold. That is, for every $i \in I$ it holds that $H'(\alpha_i) = \sum_{\ell=1}^{n} \gamma_\ell \cdot q'_\ell(\alpha_i)$.*

   (Observe that if $|I| = t$, then the above constraints fully determine $H'(x)$.)

   (b) *For every $j \notin I$, $\mathcal{S}$ simulates honest party P_j sending the value $H'(\alpha_j)$.*

5. *$\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and halts.*

We now prove that for every $I \subseteq [n]$, such that $|I| \le t$,

$$\left\{\text{IDEAL}_{F_{eval}^k,\mathcal{S}(z),I,\mathcal{S}(z),I}(\vec{\beta})\right\}_{\vec{\beta}\in\mathbb{F}^n,z\in\{0,1\}^*} \equiv \left\{\text{HYBRID}_{\pi,\mathcal{A}(z),I}^{F_{VSS}^{subshare}}(\vec{\beta})\right\}_{\vec{\beta}\in\mathbb{F}^n,z\in\{0,1\}^*}.$$

We begin by showing that the outputs of the honest parties are distributed identically in the ideal world and in the protocol execution in the $F_{VSS}^{subshare}$-hybrid model. Then, we show that the view of the corrupted parties is distributed identically, when the output of the honest parties is given. This implies that the joint distributions are identical.

The honest parties' outputs. We analyze the distribution of the output of the honest parties. Let the inputs of the honest parties be shares of the degree-t polynomial $f(x)$. Then, in the ideal world the output of the each honest party P_j is $(f(\alpha_j), f(\alpha_k))$; i.e., the output of each party is a deterministic function of the inputs of all honest parties.

In the protocol execution in the $F_{VSS}^{subshare}$ model, again when the inputs of the honest parties are shares of a degree-t polynomial $f(x)$, the $F_{VSS}^{subshare}$ functionality guarantees that the shares obtained by each party are $g_1(\alpha_j), \ldots, g_n(\alpha_j)$ where for every $\ell = 1, \ldots, n$ it holds that $g_\ell(0) = f(\alpha_\ell)$ and $\deg(g_\ell) = t$. In the protocol each honest party locally computes share of the polynomial

$$H(x) = \gamma_1 \cdot g_1(x) + \ldots + \gamma_n \cdot g_n(x),$$

for which it holds that

$$H(0) = \gamma_1 \cdot g_1(0) + \ldots \gamma_n \cdot g_n(0) = \gamma_1 \cdot f(\alpha_1) + \ldots + \gamma_n \cdot f(\alpha_n) = f(\alpha_k).$$

Thus, the $F_{VSS}^{subshare}$ functionality guarantees that the free coefficient of the polynomial $H(x)$ is the correct output. In addition, it guarantees that each polynomial $g_\ell(x)$ is of degree t, in turn implying that $H(x)$ is also of degree t. Therefore, after the parties send their shares in the final stage, each honest party holds at least $2t+1$ correct shares of the degree-t polynomial $H(x)$. This implies that the Reed-Solomon error correction procedure yields the correct codeword, irrespective of the values sent by the corrupted parties. Thus, each honest party obtains the correct polynomial $H(x)$ and outputs the value $H(0) = f(\alpha_k)$. We conclude that the exact values output by the honest parties in a protocol execution are identical to those output by the honest parties in the ideal model.

The adversary's view. We now show that the view of the adversary is identical in the hybrid and the ideal executions, given the output of the honest parties. (In fact, since the output of the honest parties is a deterministic function of their inputs, this is the same as conditioning on the inputs.) We separate the protocol into three phases: in the first phase the adversary receives shares from the $F_{VSS}^{subshare}$ functionality; in the second phase the adversary sends its polynomials to the $F_{VSS}^{subshare}$ functionality and receives back the corrected polynomials from $F_{VSS}^{subshare}$; in the third phase the adversary receives shares of the polynomial $H(x)$ as sent in the protocol. We first show that the partial view of the adversary in the first phase is distributed identically to the partial output of the simulator. Then, we show that the remaining view and the remaining output (of each phase) are distributed identically when conditioned on the partial view and output of the previous stage(s) that are given.

In the first phase of the protocol execution, the adversary receives shares from the $F_{VSS}^{subshare}$ functionality. That is, it receives $\{g_j(\alpha_i)\}_{i\in I;j\notin I}$ where each $g_j(x) \in_R \mathcal{P}^{f(\alpha_j),t}$. In the ideal world, the adversary receives from $\mathcal{S}$ simulating $F_{VSS}^{subshare}$ the shares $\left\{q'_j(\alpha_i)\right\}_{i\in I;j\notin I}$ where each $q'_j \in_R \mathcal{P}^{0,t}$. By Claim 3.4 these distributions are identical.

In the second phase, the adversary sends to $F_{VSS}^{subshare}$ a set of polynomials $\{q_i(x)\}_{i\in I}$. Since the view of the adversary is distributed identically in both worlds up until this point, the distribution of the polynomials that it sends $F_{VSS}^{subshare}$ are also identically distributed. The $F_{VSS}^{subshare}$ functionality "corrects" the adversary's polynomials, and sends them back to the adversary. This correction merely requires the functionality to check the degree of the polynomials and that their free coefficients are correct; i.e., $q_i(0) = f(\alpha_i)$. The simulator $\mathcal{S}$ carries out exactly the same check, and it can do this because it received the values $f(\alpha_i)$ for every $i \in I$ from the trusted party computing F_{eval}^k. Therefore, the values sent by $\mathcal{S}$ to $\mathcal{A}$ in the simulation are identical to those sent by the $F_{VSS}^{subshare}$ functionality to $\mathcal{A}$ in this stage of the protocol. Thus, the view of the corrupted parties up until the end of phase two of the protocol are distributed identically in both worlds.

Finally, in the last phase of the protocol, the adversary receives the shares $\{H(\alpha_j)\}_{j\notin I}$, where $H(x)$ is a polynomial that satisfies:

$$H(x) = \sum_{\ell=1}^{n} \gamma_\ell \cdot g_\ell(x) \tag{12}$$

In contrast, in the simulation, the adversary receives shares $\{H'(\alpha_j)\}_{j\notin I}$, where $H'(x)$ is a random polynomial that satisfies that $H'(0) = f(\alpha_k)$ and $H'(\alpha_i) = \sum_{\ell=1}^{n} \gamma_\ell \cdot q'_\ell(\alpha_i)$. We now show that $H(x)$ and $H'(x)$ are identically distributed. We do this by showing that for any fixed polynomial $h(x)$, and any (valid) partial view of the adversary in the first two phases together with the honest parties' output T,

$$\Pr[H(x) = h(x) \mid T] = \Pr[H'(x) = h(x) \mid T]$$

where $H(x)$ is the polynomial of Eq. (12) from the real protocol, and $H'(x)$ is the polynomial above chosen by $\mathcal{S}$ in the ideal simulation. There are three cases:

- *Case 1 – $\deg(h(x)) > t$:* In the ideal simulation, $H'(x)$ chosen by $\mathcal{S}$ is always of degree-t and so $\Pr[H'(x) = h(x) \mid T] = 0$. Likewise, in the protocol execution, $F_{VSS}^{subshare}$ guarantees that $g_1(x), \ldots, g_n(x)$ are all of degree-t and so by Eq. (12), $H(x)$ is also of degree-t. This implies that $\Pr[H(x) = h(x) \mid T] = 0$ as well.

- *Case 2 – $h(x)$ is not consistent with the points of the corrupted parties:* This case refers to the event that there exists an $i \in I$ such that $h(\alpha_i) \neq \sum_{\ell=1}^{n} \gamma_\ell \cdot g_\ell(\alpha_i)$ (in the real protocol execution), or $h(\alpha_i) \neq \sum_{\ell=1}^{n} \gamma_\ell \cdot q'_\ell(\alpha_i)$ (in the ideal simulation). Note that the points $g_\ell(\alpha_i)$ (resp., $q'_\ell(\alpha_i)$) for $i \in I$ are fully determined by the partial view T. In the ideal simulation, $H'(x)$ is chosen by $\mathcal{S}$ so that for every i, $h(\alpha_i) = \sum_{\ell=1}^{n} \gamma_\ell \cdot q'_\ell(\alpha_i)$ and so the probability is again 0. Likewise, by Eq. (12), the polynomial $H(x)$ defined in the protocol execution is always consistent and the probability is 0.

- *Case 3 – otherwise:* In this case, $\deg(h(x)) = t$, and for every $i \in I$ it holds that: $h(\alpha_i) = \sum_{\ell=1}^{n} \gamma_\ell \cdot g_\ell(\alpha_i)$ (resp., $h(\alpha_i) = \sum_{\ell=1}^{n} \gamma_\ell \cdot q'_\ell(\alpha_i)$). In the protocol execution, $F_{VSS}^{subshare}$ guarantees that the polynomials $\{g_j(x)\}_{j \notin I}$ are all random polynomials under the constraint that $g_j(0) = f(\alpha_j)$. The polynomial $H(x)$ is a linear function of all the polynomials $\{g_k(x)\}_{k=1}^{n}$. Fix $j \notin I$. By the definition of $F_{VSS}^{subshare}$, $g_j(x)$ is a random polynomial under the constraint that $g_j(0) = f(\alpha_j)$. By Corollary 3.3, given $\{g_j(\alpha_i)\}_{i \in I}$ and a fixed $g_j(0)$, it holds that for any $t - |I|$ points α_ℓ (with $\ell \notin I$) the points $g_j(\alpha_\ell)$ – which are not seen by the adversary – are uniformly distributed. This implies that for any $t - |I|$ points α_ℓ (with $\ell \notin I$) the points $H(\alpha_\ell)$ are uniformly distributed. This is identical to the way that $\mathcal{S}$ chooses $H'(x)$. ∎

6.3 The F_{VSS}^{mult} Functionality for Sharing a Product of Shares

Recall that in the semi-honest protocol for multiplication, each party locally computes the product of its *shares on the input wires* and distributes shares of this product to all other parties (i.e., it defines a polynomial with free coefficient that equals the product of its shares). In the protocol for malicious adversaries, the same procedure needs to be followed. However, in contrast to the semi-honest case, a mechanism is needed to enforce the malicious parties to indeed use the product of their shares. Note that using $F_{VSS}^{subshare}$, we can force the malicious parties to distributes shares of their shares. The functionality and protocol described in this section uses this to ensure that the malicious parties also distribute shares of the product of their shares. Stated differently, F_{VSS}^{mult} is the analog of $F_{VSS}^{subshare}$, but rather than sharing the input shares the result of the functionality is a sharing of the *product* of some party's input shares. As with $F_{VSS}^{subshare}$, for technical reasons to enable the simulation, we need to define F_{VSS}^{mult} as a reactive functionality.

FUNCTIONALITY 6.7 (The reactive F_{VSS}^{mult} functionality)

1. The F_{VSS}^{mult} functionality receives the inputs (a_j, b_j) from every honest party P_j $(j \notin I)$.
2. F_{VSS}^{mult} computes the unique degree-t polynomials A' and B' such that $A'(\alpha_j) = a_j$ and $B'(\alpha_j) = b_j$ for every $j \notin I$ (if no such A' or B' exist of degree-t, then F_{VSS}^{mult} behaves differently as in Footnote 6).
3. F_{VSS}^{mult} sends $(A'(x), B'(x))$ to the dealer P_1.
4. F_{VSS}^{mult} receives the input of the dealer P_1, which is a polynomial C or a special symbol $*$:
   (a) If the input is the special symbol $*$, then F_{VSS}^{mult} chooses a random degree-t polynomial C' under the constraint that $C'(0) = A'(0) \cdot B'(0)$.
   (b) Else, if the input is a polynomial C such that $\deg(C) = t$ and $C(0) = A'(0) \cdot B'(0)$, then F_{VSS}^{mult} sets $C' = C$.
   (c) Otherwise, if either $\deg(C) > t$ or $C(0) \neq A'(0) \cdot B'(0)$, then F_{VSS}^{mult} sets $C'(x) = A'(0) \cdot B'(0)$ to be the constant polynomial equalling $A'(0) \cdot B'(0)$ everywhere.
5. F_{VSS}^{mult} sends $C'(x)$ to the dealer P_1, and sends $(A'(\alpha_i), B'(\alpha_i), C'(\alpha_i))$ to every P_i.

The special input symbol $*$ is an instruction for the trusted party computing F_{VSS}^{mult} to choose the polynomial $C'(x)$ determining the output shares itself. This is the input used by honest parties.

We remark that although the dealing party P_1 is supposed to already have $A'(x), B'(x)$ as part of its input and each party P_i is also supposed to already have $A'(\alpha_i)$ and $B'(\alpha_i)$ as part of its input, this information is provided as output in order to enable simulation in the case that the corrupted parties use incorrect inputs.

The protocol. In the protocol we assume that the parties already hold shares of a and b (where a and b are the original shares of the dealer); as is clear from the functionality definition, their aim is to now obtain shares of $a \cdot b$. We stress that a and b are not values on the wires, but rather are the *shares* of the dealing party of the original values on the wires. Let $A(x)$ and $B(x)$ be polynomials such that $A(0) = a$ and $B(0) = b$; i.e., $A(x)$ and $B(x)$ are the polynomials used to share a and b. The idea behind the protocol is for the dealer to first define a series of t polynomials $D_1(x), \ldots, D_t(x)$, all of degree-t, such that $C(x) = A(x) \cdot B(x) - \sum_{k=1}^{t} x^k D_k(x)$ is a random degree-t polynomial with free coefficient equalling $a \cdot b$. The dealer then shares the polynomials $D_1(x), \ldots, D_t(x)$ and each party can then locally compute shares of $C(x)$. The important thing to note is that the free coefficient of $C(x)$ equals $A(0) \cdot B(0) = a \cdot b$ for *every* possible choice of polynomials $D_1(x), \ldots, D_t(x)$. This is due to the fact that each $D_k(x)$ is multiplied by x^k and so these do not affect $C(0)$. This guarantees that even if the dealer is malicious, the polynomial $C(x)$ must have the correct free coefficient.

In more detail, after defining $D_1(x), \ldots, D_t(x)$, the dealer shares them all using F_{VSS}; this ensures that all polynomials are of degree-t and all parties have correct shares. Since each party already holds a valid share of $A(x)$ and $B(x)$, this implies that each party can *locally compute* its share of $C(x)$. Specifically, given $A(\alpha_j)$, $B(\alpha_j)$ and $D_1(\alpha_j), \ldots, D_t(\alpha_j)$, party P_j can simply compute $C(\alpha_j) = A(\alpha_j) \cdot B(\alpha_j) - \sum_{k=1}^{t} (\alpha_j)^k D_k(\alpha_j)$. The crucial properties are that **(a)** if the dealer is honest, then all the honest parties hold valid shares of a random degree-t polynomial with free coefficient $a \cdot b$, as required, and **(b)** if the dealer is malicious, all honest parties are guaranteed to hold valid shares of a polynomial with free coefficient $a \cdot b$. Thus, all that remains is for the parties to verify that the shares that they hold for $C(x)$ define a degree-t polynomial.

It may be tempting to try to solve this problem by simply having the dealer share $C(x)$ using F_{VSS}, and then having each party check that the share that it received from this F_{VSS} equals the value $C(\alpha_j)$ that it computed above. Since F_{VSS} guarantees that the polynomial shared is of degree-t, and the above strategy ensures that the computed polynomial has the correct free coefficient, this seems to suffice. However, this does not work because it is possible for the dealer to define the $D_1(x), \ldots, D_t(x)$ polynomials so that $C(x)$ is a degree $2t$ polynomial that agrees with some other degree-t polynomial $C'(x)$ on up to $2t$ of the honest parties' points α_j, but for which $C'(0) \neq a \cdot b$. A malicious dealer can then share $C'(x)$ using F_{VSS} and no honest parties would detect any cheating.[8] Observe that at least one honest party would detect cheating and could complain (because $C(x)$ only agrees with $C'(x)$ on $2t$ of the points, and there are at least $2t + 1$ honest parties). However, this is not enough to act upon because when the dealer is honest, up to t of the parties could present fake complaints because they are malicious. We solve this problem by

[8]An alternative strategy could be to run the F_{VSS} protocol on the shares $C(\alpha_j)$ that the parties computed in order to verify that it is a degree-t polynomial. The problem with this strategy is that if $C(x)$ is not a degree-t polynomial, then the protocol for F_{VSS} *changes* the points that the parties receive so that it is a degree-t polynomial. However, in this process, the free coefficient of the resulting polynomial may also change. Thus, there will no longer be any guarantee that the honest parties hold shares of a polynomial with the correct free coefficient.

having the parties unequivocally verify every complaint to check if it is legitimate. If the complaint is legitimate, then they just reconstruct the initial shares a and b and all output the constant share $a \cdot b$. In contrast, if the complaint is not legitimate, the parties just ignore it. This guarantees that if no honest parties complain (legitimately), then the degree-t polynomial $C'(x)$ shared using F_{VSS} agrees with the computed polynomial $C(x)$ on at least $2t+1$ points. Since $C(x)$ if of degree at most $2t$, this implies that $C(x) = C'(x)$ and so it is actually of degree-t, as required.

In order to detect complaints, we use the new functionality defined in Section 6.2 called F_{eval} that reconstructs the input share of the complainant (given that all honest parties hold valid shares of a degree-t polynomial). Now, if a party P_j complains legitimately, then this implies that $C'(\alpha_j) \neq A(\alpha_j) \cdot B(\alpha_j) - \sum_{k=1}^{t} (\alpha_j)^k D_k(\alpha_j)$. Observe that the parties are guaranteed to have valid shares of all the polynomials $C'(x), D_1(x), \ldots, D_t(x)$, since they are shared using F_{VSS} and also of $A(x)$ and $B(x)$ by the assumption on the inputs. Thus, they can use F_{eval} to obtain all of the values $A(\alpha_j)$, $B(\alpha_j)$, $D_1(\alpha_j), \ldots, D_t(\alpha_j)$, and $C'(\alpha_j)$ and then each party can just check if $C'(\alpha_j)$ equals $A(\alpha_j) \cdot B(\alpha_j) - \sum_{k=1}^{t} (\alpha_j)^k D_k(\alpha_j)$. If yes, then the complaint is false. If no, then the complaint is valid and they reconstruct $a \cdot b$.

Building the polynomial $C(x)$. As we have mentioned above, the protocol works by having the dealer choose t polynomials $D_1(x), \ldots, D_t(x)$ with the property that $C(x) = A(x) \cdot B(x) - \sum_{k=1}^{t} x^k \cdot D_k(x)$ is a *uniformly distributed* polynomial in $\mathcal{P}^{a \cdot b, t}$, where $A(0) = a$ and $B(0) = b$. We now show how the dealer chooses these polynomials. The dealer first defines the polynomial $D(x)$ as follows:

$$D(x) \overset{\text{def}}{=} A(x) \cdot B(x) = a \cdot b + d_1 x + \ldots + d_{2t} x^{2t}.$$

Next it defines the polynomials:

$$\begin{aligned}
D_t(x) &= r_{t,0} + r_{t,1}x + \ldots + r_{t,t-1}x^{t-1} + d_{2t}x^t \\
D_{t-1}(x) &= r_{t-1,0} + r_{t-1,1}x + \ldots + r_{t-1,t-1}x^{t-1} + (d_{2t-1} - r_{t,t-1})\, x^t \\
&\vdots \\
D_1(x) &= r_{1,0} + r_{1,1}x + \ldots r_{1,t-1}x^{t-1} + (d_{t+1} - r_{i,1} - r_{i-1,2} - \ldots - r_{2,t-1})\, x^t
\end{aligned}$$

where all $r_{i,j} \in_R \mathbb{F}$ are random values, and all the d_i values are the coefficients from $D(x)$. In general, polynomial $D_k(x)$ for $1 \leq k \leq t$ is defined by:

$$D_k(x) = \sum_{\ell=0}^{t-1} r_{k,\ell} \cdot x^{\ell} + \left(d_{k+t} - \sum_{j=k+1}^{t} r_{j,t+k-j} \right) \cdot x^t$$

and the polynomial $C(x)$ is computed by:

$$C(x) = D(x) - \sum_{k=1}^{t} x^k \cdot D_k(x)$$

where $D(x) = A(x) \cdot B(x)$.

Before proceeding, we show that when the polynomials $D_1(x), \ldots, D_t(x)$ are chosen in this way, it holds that $C(x)$ is a degree-t polynomial with free coefficient $A(0) \cdot B(0) = a \cdot b$. For every polynomial $D_k(x)$, we have that: $D_k(x) = \sum_{\ell=0}^{t-1} r_{k,\ell} \cdot x^\ell + R_{k,t} \cdot x^t$, where

$$R_{k,t} = d_{k+t} - \sum_{j=k+1}^{t} r_{j,t+k-j}. \tag{13}$$

We now analyze the structure of the polynomial $\sum_{k=1}^{t} x^k \cdot D_k(x)$. First, observe that it is a polynomial of degree $2t$ with free coefficient 0. Next, the coefficient of the monomial x^ℓ is the *sum* of the coefficients of the ℓth column in Table 1; in the table, the coefficients of the polynomial $D_k(x)$ are written in the kth row and are shifted k places to the right since the polynomial $C(x)$ contains the term $x^k \cdot D_k(x)$.

	x	x^2	x^3	$\ldots$	x^t	x^{t+1}	x^{t+2}	$\ldots$	x^{2t-2}	x^{2t-1}	x^{2t}
D_t					$r_{t,0}$	$r_{t,1}$	$r_{t,2}$	$\ldots$	$r_{t,t-2}$	$r_{t,t-1}$	$R_{t,t}$
D_{t-1}				$\ldots$	$r_{t-1,1}$	$r_{t-1,2}$	$r_{t-1,3}$	$\ldots$	$r_{t-1,t-1}$	$R_{t-1,t}$	
D_{t-2}				$\ldots$	$r_{t-2,2}$	$r_{t-2,3}$	$r_{t-2,4}$	$\ldots$	$R_{t-2,t}$		
$\vdots$				$\iddots$	$\vdots$	$\vdots$	$\vdots$	$\iddots$			
D_3			$r_{3,0}$	$\ldots$	$r_{3,t-3}$	$r_{3,t-2}$	$r_{3,t-1}$	$\ldots$			
D_2		$r_{2,0}$	$r_{2,1}$	$\ldots$	$r_{2,t-2}$	$r_{2,t-1}$	$R_{2,t}$				
D_1	$r_{1,0}$	$r_{1,1}$	$r_{1,2}$	$\ldots$	$r_{1,t-1}$	$R_{1,t}$					

Table 1: Coefficients of the polynomial $\sum_{k=1}^{t} x^k \cdot D_k(x)$.

We will now show that for every $\ell = 1, \ldots, t$ the coefficient of the monomial $x^{t+\ell}$ in the polynomial $\sum_{k=1}^{t} x^k \cdot D_k(x)$ equals $d_{t+\ell}$. Now, the sum of the $(t+\ell)$th column of the above table (for $1 \leq \ell \leq t$) is

$$R_{\ell,t} + r_{\ell+1,t-1} + r_{\ell+2,t-2} + \cdots + r_{t,\ell} = R_{\ell,t} + \sum_{j=\ell+1}^{t} r_{j,t+\ell-j}$$

and so the coefficient of $x^{\ell+t}$ in the polynomial $\sum_{k=1}^{t} x^k \cdot D_k(x)$ equals $d_{t+\ell}$ if and only if

$$R_{\ell,t} = d_{\ell+t} - \sum_{j=\ell+1}^{t} r_{j,t+\ell-j},$$

which is exactly what is defined in Eq. (13). We conclude that the $(k+t)$th coefficient of the polynomial $C(x) = D(x) - \sum_{k=1}^{t} x^k \cdot D_k(x)$ equals $d_{k+t} - d_{k+t} = 0$, and thus $C(x)$ is of degree t, as required. The fact that $C(0) = a \cdot b$ follows immediately from the fact that each $D_k(x)$ is multiplied by x^k and so this does not affect the free coefficient of $D(x)$. Finally, observe that the coefficients of $x, \ldots, x^t$ are all random (since the values $r_{i,0}$ appears only in the coefficient of x^i) and so the polynomial $C(x)$ also has random coefficients everywhere except for the free coefficient.

The protocol. The protocol is implemented in the (F_{VSS}, F_{eval})-hybrid model. We assume that the dealer has already distributed its shares for the polynomials $A(x)$ and $B(x)$ using the $F_{VSS}^{subshare}$ functionality. The full description appears in Protocol 6.8.

PROTOCOL 6.8 (Securely computing F_{VSS}^{mult} in the F_{VSS}-F_{eval}-hybrid model)

- **Inputs:**
  1. Each party P_i holds a pair of shares a_i and b_i such that $a_i = A(\alpha_i)$ and $b_i = B(\alpha_i)$.
  2. The dealer has the degree-t polynomials A and B as auxiliary input.

  We assume that the above structure of the inputs holds for all honest P_j when the dealer is also honest (see the definition of F_{VSS}^{mult} above and Footnote 6 for when this does not hold).

- **Common input:** The description of a field $\mathbb{F}$ and n specified elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$.

- **The protocol:**
  1. *Dealing phase:*
     (a) The dealer P_1 defines the degree-$2t$ polynomial $D(x) = A(x) \cdot B(x)$; denote $D(x) = a \cdot b + \sum_{k=1}^{2t} d_k \cdot x^k$.
     (b) P_1 chooses t^2 values $\{r_{k,j}\}$ uniformly and independently at random from $\mathbb{F}$, where $k = 1, \ldots, t$, and $j = 0, \ldots, t-1$.
     (c) For every $k = 1, \ldots, t$, the dealer defines the polynomial $D_k(x)$:
     $$D_k(x) = \sum_{\ell=0}^{t-1} r_{k,\ell} \cdot x^\ell + \left(d_{k+t} - \sum_{j=k+1}^{t} r_{j,t+k-j} \right) \cdot x^t.$$
     (d) P_1 computes the polynomial:
     $$C(x) = D(x) - \sum_{k=1}^{t} x^k \cdot D_k(x).$$
     (e) P_1 invokes the F_{VSS} functionality as dealer with input $C(x)$; denote by $c(i)$ the output share received by party P_i.
     (f) P_1 invokes the F_{VSS} functionality as dealer with input $D_k(x)$ for every $k = 1, \ldots, t$; denote by $d_k(i)$ the output share received by party P_i.
     (g) P_1 outputs $(A(x), B(x), C(x))$.
  2. *Verify phase:* Each party P_i works as follows:
     (a) If any of $c(i), d_k(i)$ equal $\perp$ then P_i proceeds to the *reject phase* (note that if one honest party received $\perp$ then all did).
     (b) P_i computes $c'(i) = a_i \cdot b_i - \sum_{k=1}^{t} (\alpha_i)^k \cdot d_k(i)$. If $c'(i) \neq c(i)$ then P_i broadcasts $(\mathsf{complaint}, i)$.
     (c) If any party P_j broadcast $(\mathsf{complaint}, j)$ then go to the *complaint resolution phase.*
  3. *Complaint resolution phase:* Run the following for every $(\mathsf{complaint}, j)$ message:
     (a) Run $t+3$ invocations of F_{eval}^j, with party P_i inputting $a_i, b_i, c(i), d_1(i), \ldots, d_t(i)$.
     (b) Let $\tilde{A}(\alpha_j), \tilde{B}(\alpha_j), \tilde{C}(\alpha_j), \tilde{D}_1(\alpha_j), \ldots, \tilde{D}_t(\alpha_j)$ be the respective outputs from the invocations. Compute $\tilde{C}'(\alpha_j) = \tilde{A}(\alpha_j) \cdot \tilde{B}(\alpha_j) - \sum_{k=1}^{t} (\alpha_j)^k \cdot \tilde{D}_k(\alpha_j)$.
     (c) If $\tilde{C}(\alpha_j) \neq \tilde{C}'(\alpha_j)$, then proceed to the *reject phase.*
  4. *Reject phase:*
     (a) Every party P_i broadcasts a_i. Given the broadcast values $\vec{a} = (a_1, \ldots, a_n)$, where $a_j = 0$ if it was not broadcast, P_i computes $A'(x)$ to be output of Reed-Solomon decoding on $\vec{a}$.
     (b) Every party P_i broadcasts b_i. Given the broadcast values $\vec{b} = (b_1, \ldots, b_n)$, where $b_j = 0$ if it was not broadcast, P_i computes $B'(x)$ to be output of Reed-Solomon decoding on $\vec{b}$.
     (c) Every party P_i sets $c(i) = A'(0) \cdot B'(0)$.
  5. *Outputs:* Every party P_i outputs $c(i)$.

Theorem 6.9 *Let $t < n/3$. Then, Protocol 6.8 t-securely computes the F_{VSS}^{mult} functionality in the (F_{VSS}, F_{eval})-hybrid model, in the presence of a static malicious adversary.*

Proof: We separately prove the security of the protocol when the dealer is honest and when the dealer is corrupted.

Case 1 – the dealer P_1 is honest: Intuitively, in this case the simulator $\mathcal{S}$ receives the outputs $(A'(\alpha_i), B'(\alpha_i), C'(\alpha_i))$ for every $i \in I$, and simulates the view of the adversary by choosing random degree-t polynomials $D_2(x), \ldots, D_t(x)$ and defining $D_1(x)$ so that

$$\alpha_i \cdot D_1(\alpha_i) = A'(\alpha_i) \cdot B'(\alpha_i) - C'(\alpha_i) - \sum_{k=2}^{t} (\alpha_i)^k \cdot D_k(\alpha_i).$$

This computation makes sense because

$$C(x) = D(x) - \sum_{k=1}^{t} x^k \cdot D_k(x) = A(x) \cdot B(x) - x \cdot D_1(x) - \sum_{k=2}^{t} x^k \cdot D_k(x)$$

implying that

$$x \cdot D_1(x) = A(x) \cdot B(x) - C(x) - \sum_{k=2}^{t} x^k \cdot D_k(x).$$

As we will see, the polynomials $D_k(x)$ chosen by an honest dealer have the same distribution as that here (i.e., they are random under the constraint that $C(x) = A(x) \cdot B(x) - \sum_{k=1}^{t} x^k \cdot D_k(x)$). In order to simulate complaints, observe that no honest party broadcasts a $\mathsf{complaint}$. Furthermore, for every $(\mathsf{complaint}, i)$ value broadcast by $\mathcal{A}$ (for $i \in I$), the complaint resolution phase can easily be simulated since $\mathcal{S}$ knows the correct values $\tilde{A}(\alpha_i) = A'(\alpha_i)$, $\tilde{B}(\alpha_i) = B'(\alpha_i)$, $\tilde{C}(\alpha_i) = C'(\alpha_i)$. Furthermore, for every k, $\mathcal{S}$ uses $\tilde{D}_k(\alpha_i) = D_k(\alpha_i)$ as chosen initially in the simulation. We now formally describe the simulator.

The simulator $\mathcal{S}$:

1. *$\mathcal{S}$ invokes the adversary $\mathcal{A}$ with the auxiliary input z.*
2. *$\mathcal{S}$ receives from F_{VSS}^{mult} the values $(A'(\alpha_i), B'(\alpha_i), C'(\alpha_i))$ for every $i \in I$.*
3. *$\mathcal{S}$ chooses $t - 1$ random degree-t polynomials $D_2(x), \ldots, D_t(x)$.*
4. *For every $i \in I$, $\mathcal{S}$ computes:*
$$D_1(\alpha_i) = (\alpha_i)^{-1} \cdot \left(A'(\alpha_i) \cdot B'(\alpha_i) - C'(\alpha_i) - \sum_{k=2}^{t} (\alpha_i)^k \cdot D_k(\alpha_i) \right)$$
5. *$\mathcal{S}$ simulates the F_{VSS} invocations, and simulates every corrupted party P_i receiving outputs $C'(\alpha_i)$, $D_1(\alpha_i), \ldots, D_t(\alpha_i)$ from F_{VSS} in the respective invocations.*
6. *For every $i \in I$ for which $\mathcal{A}$ instructs the corrupted party P_i to broadcast a $(\mathsf{complaint}, i)$ message, $\mathcal{S}$ simulates the complaint resolution phase by simulating P_i receiving outputs $A'(\alpha_i)$, $B'(\alpha_i), C'(\alpha_i), D_1(\alpha_i), \ldots, D_t(\alpha_i)$ from the respective F_{eval} invocations.*
7. *$\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs, and halts.*

We prove that for every for every $I \subseteq [n]$, every $z \in \{0,1\}^*$ and all vectors of inputs $\vec{x}$,

$$\left\{\mathrm{IDEAL}_{F_{VSS}^{mult},\mathcal{S}(z),I}(\vec{x})\right\} \equiv \left\{\mathrm{HYBRID}_{F_{VSS}^{mult},\mathcal{A}(z),I}^{F_{VSS},F_{eval}}(\vec{x})\right\}.$$

We begin by showing that the output of the honest parties is distributed identically in the ideal world and the hybrid world. Then, we show that the view of the corrupted parties is distributed identically, when the output of the honest parties is given.

The honest parties' outputs. We analyze the distribution of the output of honest parties. Let the inputs of the honest parties be shares of the degree-t polynomials $A(x)$ and $B(x)$. Then, in the ideal model the trusted party chooses a $C'(x)$ that is distributed uniformly at random in $\mathcal{P}^{A(0)\cdot B(0),t}$, and sends each party P_j the output $(A(\alpha_j), B(\alpha_j), C'(\alpha_j))$; note that when the dealer is honest it holds that $A' = A$ and $B' = B$.

In the protocol execution in the F_{VSS}-hybrid model, we have that the honest dealer chooses $D_1(x), \ldots, D_t(x)$ as instructed. It is immediate that the polynomial C in the protocol is such that $C(0) = A(0) \cdot B(0)$ and that each honest party P_j outputs $c(j) = C(\alpha_j)$. This is due to the fact that all complaints that are broadcasted are such that $\tilde{C}(\alpha_j) = \tilde{A}(\alpha_j) \cdot \tilde{B}(\alpha_j) - \sum_{k=1}^{t} (\alpha_j)^k \cdot \tilde{D}_k(\alpha_j)$, since the dealer is honest. Thus, in the protocol execution, the honest parties output shares of a polynomial in $\mathcal{P}^{A(0)\cdot B(0),t}$. It remains to show that $C(x)$ is of degree-t and is *uniformly distributed* in $\mathcal{P}^{A(0)\cdot B(0),t}$. In the discussion above, we have already shown that $\deg(C) = t$. We now show that every coefficient of $C(x)$, except for the free coefficient, is uniformly distributed. In order to see this, observe that for every $k = 1, \ldots, t$ the value $r_{k,0}$ appears only in the polynomial $D_k(x)$. Furthermore, recall that the sum of polynomials to compute $C(x)$ includes $x^k \cdot D_k(x)$ and thus the kth coefficient of $C(x)$ includes $r_{k,0}$. Thus, fixing all other values and then choosing $r_{1,0}, \ldots, r_{t,0} \in_R \mathbb{F}$ randomly, we have that the kth coefficient is uniformly distributed (because we add the random $r_{k,0}$ to it), and so all coefficients except for the free coefficient are uniformly distributed.

We conclude that $C(x)$ as computed by the honest parties is uniformly distributed in $\mathcal{P}^{A(0)\cdot B(0),t}$ and so the distribution over the outputs of the honest parties in the real and ideal executions are identical.

The adversary's view. We now show that the view of the adversary is identical in the real protocol and ideal executions, given the honest parties' outputs. Fix the honest parties' outputs $(A'(\alpha_j), B'(\alpha_j), C'(\alpha_j))$ for every $j \notin I$. Observe that this fully determines the polynomial C' since there are more than t points (since we have already shown that the outputs of the honest parties are identically distributed we have that $\deg(C') = t$ in the protocol execution as well). (Of course, A' and B' are also fully determined, but this is anyway the case since the dealer is honest and so they are its input.)

The view of the adversary in a real protocol execution consists of the shares:

$$\left\{D_1(\alpha_i)\right\}_{i\in I}, \ldots, \left\{D_t(\alpha_i)\right\}_{i\in I}, \left\{C(\alpha_i)\right\}_{i\in I}$$

and the messages from the complaints phase.[9] The trusted party sends the simulator the points

[9]We stress that any complaints that are broadcast by corrupted parties provide no new information since the corrupted parties merely obtain the values that they already hold. Formally, since the simulator in the ideal model receives all of the values received in a complaint phase, the view obtained in the real and ideal executions are identical, as long as the $D_k(\alpha_i)$ values received are also identically distributed.

$\{C'(\alpha_i)\}_{i\in I}$, where C' is a polynomial that was chosen uniformly at random from $\mathcal{P}^{A(0)\cdot B(0),t}$ (this is due to the fact that we have assumed that the input of an honest dealer is always $C' = *$). Since C is fully determined by the honest parties' outputs (as we have seen, when the dealer is honest we have that $C = C'$), it follows that the shares obtained $\{C(\alpha_i)\}_{i\in I}$ by the honest parties in the real execution are identical to those obtained in the ideal model. It therefore suffices to show that the shares $\{D_k(\alpha_i)\}_{i\in I;k\in[t]}$ are also identically distributed, conditioned on the values $\{C(\alpha_i)\}_{i\in I}$.

We denote by $D_2^S(x), \ldots, D_t^S(x)$ the polynomials chosen by $\mathcal{S}$ in the simulation, and by $D_2(x)$, $\ldots, D_t(x)$ the polynomials chosen by the honest dealer in a protocol execution. We now show inductively that for every $j = t$ down to 2,

$$\left\{D_t^S(\alpha_i), \ldots, D_j^S(\alpha_i)\right\}_{i\in I} \equiv \left\{D_t(\alpha_i), \ldots, D_j(\alpha_i)\right\}_{i\in I} \tag{14}$$

(recall that this is conditioned on the fixed $A'(x), B'(x), C'(x)$ polynomials, but in this case of an honest dealer $A'(x) = A(x)$, $B'(x) = B(x)$, and $C'(x) = C(x)$). Consider first the case of $j = t$. In this case, we compare $\{D_t^S(\alpha_i)\}_{i\in I}$ to $\{D_t(\alpha_i)\}_{i\in I}$. By the definition of $D_t(x)$, the coefficient of x^ℓ for $\ell = 1, \ldots, t-1$ is a random value $r_{t,\ell}$ which does not appear in $C(x)$, $A(x)$ or $B(x)$. In contrast, $r_{t,0}$ is included in the computation of the tth coefficient of $C(x)$, which is fixed. Nevertheless, the tth coefficient of $C(x)$ also includes $r_{1,t-1}$ from D_1 which is uniformly distributed and so $r_{t,0}$ is independent of $C(x)$. Furthermore, the coefficient of x^t in $D_t(x)$ equals $R_{t,t} = d_{2t}$ which is the $2t$th coefficient of $A(x)\cdot B(x)$; see Table 1. However, by Corollary 3.6, any t points $\{f(\alpha_i)\}_{i\in I}$ are uniformly distributed, when f is a degree-t polynomial for which its first t coefficients are uniformly distributed. Since, $D_t^S(x)$ is chosen randomly by $\mathcal{S}$, the points $\{D_t^S(\alpha_i)\}_{i\in I}$ are also uniformly distributed.

Next, fix $\left\{D_t^S(\alpha_i), \ldots, D_{j+1}^S(\alpha_i)\right\}_{i\in I}$ and $\{D_t(\alpha_i), \ldots, D_{j+1}(\alpha_i)\}_{i\in I}$. We prove that the distribution over $\{D_j^S(\alpha_i)\}_{i\in I}$ is identical to $\{D_j(\alpha_i)\}_{i\in I}$, conditioned on $A(x), B(x), C(x)$ and on the points $\left\{D_t^S(\alpha_i), \ldots, D_{j+1}^S(\alpha_i)\right\}_{i\in I}$ and $\{D_t(\alpha_i), \ldots, D_{j+1}(\alpha_i)\}_{i\in I}$, respectively. Abusing notation, we write:

$$\begin{aligned}&\left\{D_j^S(\alpha_i) \;\middle|\; D_t^S(\alpha_i), \ldots, D_{j+1}^S(\alpha_i), A(x), B(x), C(x)\right\}_{i\in I}\\ &\qquad\equiv \left\{D_j(\alpha_i) \;\middle|\; D_t(\alpha_i), \ldots, D_{j+1}(\alpha_i), A(x), B(x), C(x)\right\}_{i\in I}\end{aligned}$$

It is clear that the points $\{D_j^S(\alpha_i)\}_{i\in I}$ are uniformly distributed, because $\mathcal{S}$ chooses $D_j^S(x)$ uniformly at random (and independently of $D_t^S(x), \ldots, D_{j+1}^S(x), A(x), B(x), C(x)$). In contrast, there seems to be dependence between $D_j(x)$ and $D_t(x), \ldots, D_{j+1}(x), A(x), B(x), C(x)$. First, regarding $A(x), B(x), C(x)$, the polynomial $D_j(x)$ is independent because of the coefficients of D_1, as mentioned above for the case of $j = t$. Regarding $D_t(x), \ldots, D_{j+1}(x)$, observe that the ℓth coefficient ($0 \le \ell \le t-1$) of $D_j(x)$ is $r_{j,\ell}$ and this is not dependent on any of the polynomials $D_{j+1}(x), \ldots, D_t(x)$. In order to see this, recall that $R_{k,t}$ depends only on $r_{j,\zeta}$ values with $j > k$ (see Table 1 and Eq. (13)). Thus, the $R_{k,t}$ values for $k = j+1, \ldots, t$ do not contain the values $r_{j,\ell}$ at all. In contrast, the tth coefficient is fully determined by the polynomials $D_{j+1}(x), \ldots, D_t(x), A(x), B(x)$. Nevertheless, by once again applying Corollary 3.6, we have that the points received by the adversary are uniformly distributed. We therefore conclude that Eq. (14) holds.

It remains now to show that the points $\{D_1^S(\alpha_i)\}_{i\in I}$ and $\{D_1(\alpha_i)\}_{i\in I}$ are also identically distributed, conditioned on all the other points $\{D_k(\alpha_i)\}_{i\in I}$ and the polynomial $C(x)$. However, the polynomial D_1 chosen by the dealer in the real protocol is fully determined by $C(x)$ and $D_2(x),\ldots,D_t(x)$. Indeed, an equivalent way of describing the dealer is for it to choose all $D_2(x),\ldots,D_t(x)$ as before, to choose $C(x)$ uniformly at random in $\mathcal{P}^{a\cdot b,t}$ and then to choose $D_1(x)$ so that

$$x\cdot D_1(x) = A(x)\cdot B(x) - C(x) - \sum_{k=2}^{t} x^k \cdot D_k(x).$$

(In order to see that this is true, observe that the tth coefficient of $D_1(x)$ is fully determined by this equation because it must equal the $t+1$th coefficient of $A(x)\cdot B(x) - C(x) - \sum_{k=2}^{t} x^k \cdot D_k(x)$, and $C(x)$ is only of degree-t so has no influence on this. Then, choosing all the other coefficients of $D_1(x)$ at random is equivalent to choosing $C(x)$ at random from $\mathcal{P}^{a\cdot b,t}$.) Thus, once $D_2(x),\ldots,D_t(x),A(x),B(x),C(x)$ are fixed, the polynomial $D_1(x)$ is fully determined. Likewise, in the simulation, the points $\{D_1(\alpha_i)\}_{i\in I}$ are fully determined by $\{D_2(\alpha_i),\ldots,D_t(\alpha_i),A(\alpha_i),B(\alpha_i),C(\alpha_i)\}_{i\in I}$, by the same equation as in the real protocol execution.

We conclude that the view of the corrupted parties in the protocol is identically distributed to the adversary's view in the ideal simulation, given the outputs of the honest parties. Combining this with the fact that the outputs of the honest parties are identically distributed in the real and ideal executions, we conclude that the joint distributions of the adversary's output and the honest parties' outputs in the ideal and real executions are identical.

Case 2 – the dealer is corrupted: Intuitively, security holds in this case because (a) the dealer receives no messages from the honest parties (unless there are complaints in which case it learns nothing it did not know), and (b) any deviation by a corrupted dealer from the prescribed instructions is detected in the verify phase. We now formally describe the simulator.

The simulator $\mathcal{S}$:

1. *$\mathcal{S}$ invokes $\mathcal{A}$ with the auxiliary input z.*
2. *$\mathcal{S}$ receives $(A'(x),B'(x))$ from F_{VSS}^{mult}.*
3. *$\mathcal{S}$ receives the polynomials $C(x)$, $D_1(x),\ldots,D_t(x)$ that $\mathcal{A}$ instructs the corrupted dealer to use in the F_{VSS} invocations.*
4. *If $\deg(C) > t$ or if $\deg(D_k) > t$ for some k, then $\mathcal{S}$ proceeds to Step 8 below.*
5. *For every $j\notin I$ such that $C(\alpha_j) \neq A'(\alpha_j)\cdot B'(\alpha_j) - \sum_{k=1}^{t}(\alpha_j)^k\cdot D_k(\alpha_j)$, simulator $\mathcal{S}$ simulates P_j broadcasting* (complaint, j) *and simulates the complaint resolution phase. In this phase, $\mathcal{S}$ uses the polynomials $A'(x),B'(x),C(x)$ and $D_1(x),\ldots,D_k(x)$ in order to compute the values received from the F_{eval} invocations. If there exists such a $j\notin I$ as above, then $\mathcal{S}$ proceeds to Step 8 below.*
6. *For every* (complaint, i) *message that was broadcast by a corrupted party P_i, simulator $\mathcal{S}$ generates the results of the F_{eval} executions. Then, if there exists an $i\in I$ such that $C(\alpha_i)\neq A'(\alpha_i)\cdot B'(\alpha_i) - \sum_{k=1}^{t}(\alpha_i)^k\cdot D_k(\alpha_i)$, simulator $\mathcal{S}$ proceeds to Step 8 below.*

7. *If $\mathcal{S}$ reaches this point, then it sends $C(x)$, obtained from $\mathcal{A}$ above, to F_{VSS}^{mult}. It then skips to Step 9 below.*

8. Simulating reject:

   (a) *$\mathcal{S}$ sends $C(x) = x^{t+1}$ to the trusted party computing F_{VSS}^{mult} (i.e., $\mathcal{S}$ sends a polynomial C such that* $\deg(C) > t$*).*

   (b) *$\mathcal{S}$ receives $C'(x)$ from F_{VSS}^{mult}, simulates every honest party P_j broadcasting $a_j = A'(\alpha_j)$ and $b_j = B'(\alpha_j)$ as in the reject phase.*

9. *$\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs, and halts.*

The simulator obtains $A'(x), B'(x)$ from F_{VSS}^{mult} and therefore can compute the actual inputs $a_j = A'(\alpha_j)$ and $b_j = B'(\alpha_j)$ held by all honest parties P_j ($j \notin I$). Therefore, the view of the adversary in the simulation is clearly identical to its view in a real execution. We now show that the output of the honest parties in the ideal model, and in the hybrid model are identical, *given* the view of the corrupted parties/adversary. We have two cases:

1. *Case 1 – no reject:* This case occurs if (a) all the polynomials $C(x)$, $D_1(x), \ldots, D_t(x)$ are of degree t, (b) it holds that $C(\alpha_j) = A'(\alpha_j) \cdot B'(\alpha_j) - \sum_{k=1}^{t} (\alpha_j)^k \cdot D_k(\alpha_j)$ for every $j \notin I$, and (c) if any corrupt P_i broadcast $(\mathsf{complaint}, i)$ then $C(\alpha_i) = A'(\alpha_i) \cdot B'(\alpha_i) - \sum_{k=1}^{t} (\alpha_i)^k \cdot D_k(\alpha_i)$. In this case, each honest party P_j in the real protocol execution outputs $C(\alpha_j)$. In the ideal model, each honest P_j also outputs $C(\alpha_j)$ as long as $\deg(C) = t$ and $C(0) = A'(0) \cdot B'(0)$. Now, let $C'(x) = A'(x) \cdot B'(x) - \sum_{k=1}^{t} x^k \cdot D_k(x)$. By the definition of C' and the fact that each $D_k(x)$ is guaranteed to be of degree-t, we have that $C'(x)$ is of degree at most $2t$. Furthermore, in this case, $C(x) = C'(x)$ on at least $2t + 1$ points $\{\alpha_j\}_{j \notin I}$. Therefore, $C(x) = C'(x)$ on all points, and in particular $C(0) = C'(0)$. Thus, we conclude that C is a degree-t polynomial such that $C(0) = A'(0) \cdot B'(0)$, and so every honest party P_j outputs $C(\alpha_j)$ in the ideal model.

2. *Case 2 – reject:* This case occurs if any of (a), (b) or (c) above do not hold. When this occurs, all honest parties run the reject phase in the real execution and output the value $A'(0) \cdot B'(0)$. In the ideal model, in any of these cases the simulator $\mathcal{S}$ sends the polynomial $C(x) = x^{t+1}$ to F_{VSS}^{mult}. Upon input of $C(x)$ with $\deg(C) > t$, functionality F_{VSS}^{mult} sets $C'(x) = A'(0) \cdot B'(0)$ and so all honest parties output the value $A'(0) \cdot B'(0)$, exactly as in the real execution.

This concludes the proof. ■

6.4 The F_{mult} Functionality and its Implementation

We are finally ready to show how to securely compute the product of shared values, in the presence of malicious adversaries.

The functionality. We begin by defining the multiplication functionality for the case of malicious adversaries. In the semi-honest setting, the F_{mult} functionality was defined as follows:

$$F_{mult}\Big((f_a(\alpha_1), f_b(\alpha_1)), \ldots, (f_a(\alpha_n), f_b(\alpha_n))\Big) = \Big(f_{ab}(\alpha_1), \ldots, f_{ab}(\alpha_n)\Big)$$

where f_{ab} is a random polynomial with free coefficient $f_a(0) \cdot f_b(0) = a \cdot b$. We stress that unlike in F_{VSS}^{mult}, here the values a and b are the actual values on the incoming wires to the multiplication gate.

In the malicious setting, we need to define the functionality with more care. First, the corrupted parties are able to influence the output and determine up to $|I|$ of the points of the output polynomial. This is due to the fact that in the protocol, the corrupted parties may choose the polynomials that they use in F_{VSS}^{mult} (for sharing the product of their shares), after receiving the shares of the products from the honest parties. In addition, as with F_{eval}, the simulator needs to receive the correct shares of the corrupted parties in order to simulate, and so this is also received as output. Since this information is anyway given to the corrupted parties, this makes no difference to the use of the functionality for secure computation. Due to the above, we define the F_{mult} multiplication functionality as a reactive functionality:

FUNCTIONALITY 6.10 (The reactive F_{mult} functionality)

1. The F_{mult} functionality receives the inputs of the honest parties $\{(\beta_j, \gamma_j)\}_{j \notin I}$. Let $f_a(x), f_b(x)$ be the unique degree-t polynomials determined by the points $\{(\alpha_j, \beta_j)\}_{j \notin I}$, $\{(\alpha_j, \gamma_j)\}_{j \notin I}$, respectively.

   (If such polynomials do not exist then no security is guaranteed; see Footnote 6.)
2. F_{mult} sends $\{(f_a(\alpha_i), f_b(\alpha_i))\}_{i \in I}$ to the (ideal) adversary.
3. F_{mult} receives points $\{\delta_i\}_{i \in I}$ from the (ideal) adversary.
4. F_{mult} chooses a random degree-t polynomial $f_{ab}(x)$ under the constraints that:
   (a) $f_{ab}(0) = f_a(0) \cdot f_b(0)$, and
   (b) For every $i \in I$, $f_{ab}(\alpha_i) = \delta_i$.

   (such a degree-t polynomial always exists since $|I| \leq t$).
5. The functionality F_{mult} sends the value $f_{ab}(\alpha_j)$ to every honest party P_j ($j \notin I$).

We are now ready to show how to multiply in the $F_{VSS}^{subshare}$ and F_{VSS}^{mult} hybrid model. Intuitively, the parties first distribute shares of their shares and shares of the product of their shares, using $F_{VSS}^{subshare}$ and F_{VSS}^{mult}, respectively. Next, we use the method from [16] to have the parties directly compute shares of the product of the values on the input wires. This method is based on the following observation. Let $f_a(x)$ and $f_b(x)$ be two degree-t polynomials such that $f_a(0) = a$ and $f_b(0) = b$, and let $h(x) = f_a(x) \cdot f_b(x) = ab + h_1 \cdot x + h_2 \cdot x^2 + \ldots + h_{2t} \cdot x^{2t}$. Letting $V_{\vec{\alpha}}$ be the Vandermonde matrix for $\vec{\alpha}$, and recalling that $V_{\vec{\alpha}}$ is invertible, we have that

$$V_{\vec{\alpha}} \cdot \begin{pmatrix} ab \\ h_1 \\ \vdots \\ h_{2t} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} h(\alpha_1) \\ h(\alpha_2) \\ \vdots \\ h(\alpha_n) \end{pmatrix} \quad \text{and so} \quad \begin{pmatrix} ab \\ h_1 \\ \vdots \\ h_{2t} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = V_{\vec{\alpha}}^{-1} \cdot \begin{pmatrix} h(\alpha_1) \\ h(\alpha_2) \\ \vdots \\ h(\alpha_n) \end{pmatrix}.$$

Let $\lambda_1, \ldots, \lambda_n$ be the first row of $V_{\vec{\alpha}}^{-1}$. It follows that

$$ab = \lambda_1 \cdot h(\alpha_1) + \ldots + \lambda_n \cdot h(\alpha_n) = \lambda_1 \cdot f_a(\alpha_1) \cdot f_b(\alpha_1) + \ldots + \lambda_n \cdot f_a(\alpha_n) \cdot f_b(\alpha_n).$$

We conclude that the parties simply need to compute a linear function of their inputs. Using $F_{VSS}^{subshare}$ and F_{VSS}^{mult}, as described above, the parties first distribute random shares of the values $f_a(\alpha_i) \cdot f_b(\alpha_i)$, for every $i = 1, \ldots, n$. That is, let $C_1(x), \ldots, C_n(x)$ be random degree-t polynomials such that for every i it holds that $C_i(0) = f_a(\alpha_i) \cdot f_b(\alpha_i)$. Then, the result of the sharing via F_{VSS}^{mult} is that each party P_i holds $C_1(\alpha_i), \ldots, C_n(\alpha_i)$. Thus, each P_i can locally compute $H(\alpha_i) = \sum_{\ell=1}^{n} \lambda_\ell \cdot C_\ell(\alpha_i)$ and we have that the parties hold shares of the polynomial $H(x) = \sum_{\ell=1}^{n} \lambda_\ell \cdot C_\ell(x)$. By the fact that $C_i(0) = f_a(\alpha_i) \cdot f_b(\alpha_i)$ for every i, it follows that

$$H(0) = \sum_{\ell=1}^{n} \lambda_\ell \cdot C_\ell(0) = \sum_{\ell=1}^{n} \lambda_\ell \cdot f_a(\alpha_i) \cdot f_b(\alpha_i) = ab. \tag{15}$$

Furthermore, since all the $C_\ell(x)$ polynomials are of degree-t, the polynomial $H(x)$ is also of degree-t. Full details of the protocol are given in Protocol 6.11.

PROTOCOL 6.11 (Computing F_{mult} in the $(F_{VSS}^{subshare}, F_{VSS}^{mult})$-hybrid model)

- **Input:** Each party P_i holds a_i, b_i, where $a_i = f_a(\alpha_i)$, $b_i = f_b(\alpha_i)$ for some polynomials $f_a(x), f_b(x)$ with degree t, which hide a, b, respectively. (If not all the points lie on a single degree-t polynomial, then no security guarantees are obtained. See Footnote 6.)
- **Common input:** The description of a field $\mathbb{F}$ and n specified elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$. In addition, the parties have constants $\lambda_1, \ldots, \lambda_n$ which are the first row of the matrix $V_{\vec{\alpha}}^{-1}$.
- **The protocol:**
  1. The parties invoke the $F_{VSS}^{subshare}$ functionality with each party P_i using a_i as its private input. Each party P_i receives back shares $A_1(\alpha_i), \ldots, A_n(\alpha_i)$, and a polynomial $A_i(x)$. (Recall that for every i, the polynomial $A_i(x)$ is of degree-t and $A_i(0) = f_a(\alpha_i)$.)
  2. The parties invoke the $F_{VSS}^{subshare}$ functionality with each party P_i using b_i as its private input. Each party P_i receives back shares $B_1(\alpha_i), \ldots, B_n(\alpha_i)$, and a polynomial $B_i(x)$.
  3. For every $i = 1, \ldots, n$, the parties invoke the F_{VSS}^{mult} functionality as follows:
     (a) *Inputs:* In the ith invocation, party P_i plays the dealer and sends F_{VSS}^{mult} the special input symbol $*$. In addition, all parties P_j $(1 \le j \le n)$ send F_{VSS}^{mult} their shares $A_i(\alpha_j), B_i(\alpha_j)$.
     (b) *Outputs:* P_i receives back the polynomial $C_i(x)$ as output (recall that $C_i(x) \in_R \mathcal{P}^{A_i(0)\cdot B_i(0),t}$. Every party P_j $(1 \le j \le n)$ receives back the value $C_i(\alpha_j)$.
  4. At this stage, each party P_i holds values $C_1(\alpha_i), \ldots, C_n(\alpha_i)$, and locally computes $H(\alpha_i) = \sum_{j=1}^{n} \lambda_j \cdot C_j(\alpha_i)$ and outputs this value.

More efficient multiplication [1]. The protocol that we have presented is very close to that described by BGW. However, it is possible to use these techniques to achieve a more efficient multiplication protocol. For example, observe that if the parties already hold shares of all other parties' shares, then these can be used directly in F_{VSS}^{mult} without running $F_{VSS}^{subshare}$ at all. Now, the verifiable secret sharing protocol of [4] presented here is based on bivariate polynomials, and so all parties do indeed receive shares of all other parties' shares. This means that it is possible to modify Protocol 6.11 so that the parties proceed directly to F_{VSS}^{mult} without using $F_{VSS}^{subshare}$ at all. Furthermore, the output of each party P_i in F_{VSS}^{mult} is the share $c(i)$ received via the F_{VSS} functionality; see Protocol 6.8. Once again, using VSS based on bivariate polynomials, this means that the parties can actually output the shares of all other parties' shares as well. Applying the linear computation of $H(x)$ to these bivariate shares, we conclude that it is possible to include the shares of all other parties as additional output from Protocol 6.11. Thus, the next time that F_{mult} is called, the parties will again already have the shares of all other parties' shares and $F_{VSS}^{subshare}$ need not be called. This is a significant efficiency improvement. (Note that unless some of the parties behave maliciously, F_{VSS}^{mult} itself requires $t+1$ invocations of F_{VSS} and nothing else. With this efficiency improvement, we have that the entire cost of F_{mult} is $n \cdot (t+1)$ invocations of F_{VSS}.) See [1] for more details on this and other ways to further utilize the properties of bivariate secret sharing in order to obtain simpler and much more efficient multiplication protocols.

Theorem 6.12 *Let $t < n/3$. Then, Protocol 6.11 t-securely computes the F_{mult} functionality in the $(F_{VSS}^{subshare}, F_{VSS}^{mult})$-hybrid model, in the presence of a static malicious adversary.*

Proof: As we have mentioned, in our analysis here we assume that the inputs of the honest parties all lie on two polynomials of degree t; otherwise (vacuous) security is immediate as described in Footnote 6. We have already discussed the motivation behind the protocol and therefore proceed directly to the simulator.

The simulator $\mathcal{S}$.

1. *$\mathcal{S}$ invokes $\mathcal{A}$ with the auxiliary input z.*
2. *$\mathcal{S}$ receives from the trusted party computing F_{mult} the values $(f_a(\alpha_i), f_b(\alpha_i))$, for every $i \in I$.*
3. *$\mathcal{S}$* simulates the first invocation of $F_{VSS}^{subshare}$:
   (a) *For every $j \notin I$, $\mathcal{S}$ chooses a uniformly distributed polynomial $A'_j(x) \in_R \mathcal{P}^{0,t}$.*
   (b) *$\mathcal{S}$ hands $\mathcal{A}$ the values $\{A'_j(\alpha_i)\}_{j \notin I; i \in I}$ as if coming from $F_{VSS}^{subshare}$ (see Step 2 of the functionality definition).*
   (c) *$\mathcal{S}$ receives from $\mathcal{A}$ a set of polynomials $\{A'_i(x)\}_{i \in I}$. If any polynomial is missing, then $\mathcal{S}$ sets it to be the constant polynomial 0.*
   (d) *For every $i \in I$, $\mathcal{S}$ performs the following checks (as checked by $F_{VSS}^{subshare}$):*
      i. *$\mathcal{S}$ checks that $A'_i(0) = f_a(\alpha_i)$, and*
      ii. *$\mathcal{S}$ checks that the degree of $A'_i(x)$ is t.*

      If either of these checks fail, $\mathcal{S}$ sets $A'_i(x)$ to be the constant polynomial that equals $f_a(\alpha_i)$ everywhere (recall that $\mathcal{S}$ received $f_a(\alpha_i)$ from F_{mult} and so can carry out this check and set the output to be these values if necessary).
   (e) *$\mathcal{S}$ hands $\mathcal{A}$ the polynomials $\{A'_i(x)\}_{i \in I}$ and the shares $\{A'_1(\alpha_i), \ldots, A'_n(\alpha_i)\}_{i \in I}$ as if coming from $F_{VSS}^{subshare}$.*

4. $\mathcal{S}$ simulates the second invocation of $F_{VSS}^{subshare}$: *This simulation is carried out in an identical way using the points $\{f_b(\alpha_i)\}_{i\in I}$. Let $B'_1(x),\ldots,B'_n(x)$ be the resulting polynomials held by $\mathcal{S}$ after the simulation of this step.*

   (Note that at this point $\mathcal{S}$ holds a set of degree-t polynomials $\{A'_k(x),B'_k(x)\}_{k\in[n]}$, where for every $j\notin I$ it holds that $A'_j(0)=B'_j(0)=0$, and for every $i\in I$ it holds that $A'_i(0)=f_a(\alpha_i)$ and $B'_i(0)=f_b(\alpha_i)$.)

5. For every $j\notin I$, $\mathcal{S}$ simulates the F_{VSS}^{mult} invocation where the honest party P_j is dealer:
   (a) *$\mathcal{S}$ chooses a uniformly distributed polynomial $C'_j(x)\in_R\mathcal{P}^{0,t}$, and hands $\mathcal{A}$ the values $\{A'_j(\alpha_i),B'_j(\alpha_i),C'_j(\alpha_i)\}_{i\in I}$ as if coming from F_{VSS}^{mult}.*

6. For every $i\in I$, $\mathcal{S}$ simulates the F_{VSS}^{mult} invocation where the corrupted party P_i is dealer:
   (a) *$\mathcal{S}$ hands the adversary $\mathcal{A}$ the polynomials $(A'_i(x),B'_i(x))$ as if coming from F_{VSS}^{mult}.*
   (b) *$\mathcal{S}$ receives from $\mathcal{A}$ the input that $\mathcal{A}$ sends to F_{VSS}^{mult}.*
      i. *If the input is the special symbol $*$, then $\mathcal{S}$ chooses a random $C'_i(x)\in_R\mathcal{P}^{f_a(\alpha_i)\cdot f_b(\alpha_i),t}$.*
      ii. *If the input is a polynomial C_i such that $\deg(C_i)=t$ and $C_i(0)=A'_i(0)\cdot B'_i(0)=f_a(\alpha_i)\cdot f_b(\alpha_i)$, then $\mathcal{S}$ sets $C'_i(x)=C_i(x)$.*
      iii. *Otherwise, $\mathcal{S}$ sets $C'_i(x)$ to be the constant polynomial equalling $f_a(\alpha_i)\cdot f_b(\alpha_i)$ everywhere.*
   (c) *$\mathcal{S}$ hands $\mathcal{A}$ the polynomial $C'_i(x)$ and shares $\{(A'_i(\alpha_k),B'_i(\alpha_k),C'_i(\alpha_k))\}_{k\in I}$, as if coming from F_{VSS}^{mult}.*

7. *For every $i\in I$, the simulator $\mathcal{S}$ computes $H(\alpha_i)=\sum_{\ell=1}^{n}\lambda_\ell\cdot C'_\ell(\alpha_i)$, where $C'_1(x),\ldots,C'_n(x)$ are as determined by $\mathcal{S}$ above, and sends the set $\{H(\alpha_i)\}_{i\in I}$ to the F_{mult} functionality (this is the set $\{\delta_i\}_{i\in I}$ in Step 3 of the functionality definition).*

8. *$\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.*

Observe that the only difference between the simulation with $\mathcal{S}$ and $\mathcal{A}$, and an execution of Protocol 6.11 with $\mathcal{A}$, is due to the fact that for every $j\notin I$, $\mathcal{S}$ chooses the polynomials $A'_j(x),B'_j(x)$, and $C'_j(x)$ to have free coefficients of 0 instead of free coefficients $f_a(\alpha_j),f_b(\alpha_j)$, and $f_a(\alpha_j)\cdot f_b(\alpha_j)$, respectively. We stress that apart from this, the executions are identical since $\mathcal{S}$ is able to run the checks of the $F_{VSS}^{subshare}$ and F_{VSS}^{mult} functionalities exactly as they are specified. Our proof proceeds by first constructing an alternative simulator that is the same as $\mathcal{S}$ except that it has the honest parties' inputs and uses them to choose $A'_j(x),B'_j(x)$, and $C'_j(x)$ like the honest parties. Then, we show that this alternative simulation is the same as the original one, since $|I|$ points on these polynomials are distributed identically in both cases. Finally, we show that the alternative simulation is identical to a real protocol execution, concluding the proof.

An alternative simulator. Let $\mathcal{S}'$ be exactly the same as $\mathcal{S}$, except that it receives for input the values $f_a(\alpha_j),f_b(\alpha_j)$, for every $j\notin I$. Then, instead of choosing $A'_j(x)\in_R\mathcal{P}^{0,t}$, $B'_j(x)\in_R\mathcal{P}^{0,t}$, and $C'_j(x)\in_R\mathcal{P}^{0,t}$, the alternative simulator $\mathcal{S}'$ chooses $A'_j(x)\in_R\mathcal{P}^{f_a(\alpha_j),t}$, $B'_j(x)\in_R\mathcal{P}^{f_b(\alpha_j),t}$, and $C'_j(x)\in_R\mathcal{P}^{f_a(\alpha_j)\cdot f_b(\alpha_j),t}$. We stress that $\mathcal{S}'$ runs in the ideal model with the same trusted party running F_{mult} as $\mathcal{S}$, and the honest parties receive output as specified by F_{mult} when running with the ideal adversary $\mathcal{S}$ or $\mathcal{S}'$.

The original and alternative simulations. We begin by showing that the joint output of the adversary and honest parties is identical in the original and alternative simulations. That is,

$$\left\{\text{IDEAL}_{F_{mult},\mathcal{S}(z),I}(\vec{x})\right\}_{\vec{x}\in(\{0,1\}^*)^n,z\in\{0,1\}^*} \equiv \left\{\text{IDEAL}_{F_{mult},\mathcal{S}'(z'),I}(\vec{x})\right\}_{\vec{x}\in(\{0,1\}^*)^n,z\in\{0,1\}^*}$$

where z' contains the same z as $\mathcal{A}$ receives, together with the $f_a(\alpha_j), f_b(\alpha_j)$ values. In order to see that the above holds, observe that both $\mathcal{S}$ and $\mathcal{S}'$ can work when given the points $\{A'_j(\alpha_i), B'_j(\alpha_i), C'_j(\alpha_i)\}_{i\in I;j\notin I}$, and they don't actually need the polynomials themselves. Furthermore, the only difference between $\mathcal{S}$ and $\mathcal{S}'$ is whether these polynomials are chosen with zero free coefficients, or with the "correct" ones. That is, there exists a machine $\mathcal{T}$ that receives points $\{A'_j(\alpha_i), B'_j(\alpha_i), C'_j(\alpha_i)\}_{i\in I;j\notin I}$ and runs the simulation strategy with $\mathcal{A}$ while interacting with F_{mult} in an ideal execution, such that:

- If $A'_j(0) = B'_j(0) = C'_j(0) = 0$ then the joint output of $\mathcal{T}$ and the honest parties in the ideal execution is exactly that of $\text{IDEAL}_{F_{mult},\mathcal{S}(z),I}(\vec{x})$; i.e., an ideal execution with the original simulator.
- If $A'_j(0) = f_a(\alpha_j)$, $B'_j(0) = f_b(\alpha_j)$ and $C'_j(0) = f_a(\alpha_j)\cdot f_b(\alpha_j)$ then the joint output of $\mathcal{T}$ and the honest parties in the ideal execution is exactly that of $\text{IDEAL}_{F_{mult},\mathcal{S}'(z'),I}(\vec{x})$; i.e., an ideal execution with the alternative simulator.

By Claim 3.4, the points $\{A'_j(\alpha_i), B'_j(\alpha_i), C'_j(\alpha_i)\}_{i\in I;j\notin I}$ when $A'_j(0) = B'_j(0) = C'_j(0) = 0$ are identically distributed to the points $\{A'_j(\alpha_i), B'_j(\alpha_i), C'_j(\alpha_i)\}_{i\in I;j\notin I}$ when $A'_j(0) = f_a(\alpha_j)$, $B'_j(0) = f_b(\alpha_j)$ and $C'_j(0) = f_a(\alpha_j)\cdot f_b(\alpha_j)$. Thus, the joint outputs of the adversary and honest parties in both simulations must be identical.

The alternative simulation and a protocol execution. We now proceed to show that the joint output of the adversary and honest parties are identical in a protocol execution and in the alternative simulation.:

$$\left\{\text{IDEAL}_{F_{mult},\mathcal{S}'(z'),I}(\vec{x})\right\}_{\vec{x}\in(\{0,1\}^*)^n,z\in\{0,1\}^*} \equiv \left\{\text{HYBRID}^{F_{VSS}^{subshare},F_{VSS}^{mult}}_{\pi,\mathcal{A}(z),I}(\vec{x})\right\}_{\vec{x}\in(\{0,1\}^*)^n,z\in\{0,1\}^*}.$$

In order to see this, we compare the ideal execution of $\mathcal{S}'$ with F_{mult} to an ideal execution of yet another simulator $\hat{\mathcal{S}}$ with a new functionality $\hat{F}_{mult}$, defined as follows:

- $\hat{F}_{mult}$ is the same as F_{mult} except that instead of receiving points $\{\delta_i\}_{i\in I}$ from the ideal adversary $\hat{\mathcal{S}}$, it receives polynomials $C'_1(x),\ldots,C'_n(x)$ from $\hat{\mathcal{S}}$. It then defines $\hat{f}_{ab}(x) = \sum_{\ell=1}^{n}\lambda_\ell\cdot C'_\ell(x)$, and gives party P_j the output $\hat{f}_{ab}(\alpha_j)$; recall that in F_{mult} party P_j receives $f_{ab}(\alpha_j)$ as output.
- $\hat{\mathcal{S}}$ is the same as $\mathcal{S}'$ except that instead of sending points $\{\delta_i = H(\alpha_i)\}_{i\in I}$ to the trusted party, it sends the polynomials $C'_1(x),\ldots,C'_n(x)$ that it defined in the (alternative) simulation.

We stress that this modification of F_{mult} to $\hat{F}_{mult}$ is merely a mental experiment for analyzing the specific alternative simulator $\mathcal{S}'$. It is immediate that the joint output of $\hat{\mathcal{S}}$ and the honest parties in an ideal execution with $\hat{F}_{mult}$ is identically distributed to the joint output of $\mathcal{A}$ and the honest parties in a real protocol execution. This is due to the fact that $\mathcal{S}'$ defines the polynomials $A'_\ell(x), B'_\ell(x), C'_\ell(x)$ identically to the way that they are chosen by $F_{VSS}^{subshare}$ and F_{VSS}^{mult} in a real protocol execution, and due to the fact that the output of the honest parties is defined using the same linear function H of the polynomials $C'_\ell(x)$ in both cases.

It thus remains to show that the joint output of $\mathcal{S}'$ and the honest parties in an ideal execution with F_{mult} is identically distributed to the joint output of $\hat{\mathcal{S}}$ and the honest parties in an ideal execution with $\hat{F}_{mult}$. First, for every $i \in I$, define $\hat{\delta}_i = \sum_{\ell=1}^{n} \lambda_\ell \cdot C'_\ell(\alpha_i)$ in the execution of $\hat{F}_{mult}$ with $\hat{\mathcal{S}}$. By the way that $\mathcal{S}'$ defines the points $\{\delta_i\}_{i \in I}$, we have that $\hat{\delta}_i = \delta_i$ for every $i \in I$. Now, $\hat{f}_{ab}(\alpha_i) = \hat{\delta}_i$ (in $\hat{F}_{mult}$ with $\hat{\mathcal{S}}$) and $f_{ab}(\alpha_i) = \delta_i$ (in F_{mult} with $\mathcal{S}'$). Thus, the polynomials f_{ab} and $\hat{f}_{ab}$ agree on all the points $\{\alpha_i\}_{i \in I}$. In addition, $f_{ab}(0) = \hat{f}_{ab}(0)$ since they both equal $f_a(0) \cdot f_b(0)$; for, $f_{ab}(0)$ this is by the way it is chosen, and for $\hat{f}_{ab}(0)$ this is by the property of H as shown in Eq. (15). If $|I| = t$ then $\hat{f}_{ab}(x) = f_{ab}(x)$ because $t+1$ points fully define a degree-t polynomial. Otherwise, the difference between them is that F_{mult} chooses $f_{ab}(x)$ randomly under the constraint that $f_{ab}(0) = f_a(0) \cdot f_b(0)$ and $f_{ab}(\alpha_i) = \delta_i$ for every $i \in I$. Observe that this is equivalent to choosing $t - |I|$ values $\beta_\ell \in_R \mathbb{F}$ at random (with $\ell \notin I$), and setting f_{ab} to be the unique polynomial such that $f_{ab}(\alpha_\ell) = \beta_\ell$ in addition to the above constraints. In contrast, $\hat{F}_{mult}$ defines $f_{ab}(x) = H(x) = \sum_{\ell=1}^{n} \lambda_\ell \cdot C'_\ell(x)$. Fix $j \notin I$. By the way $\hat{\mathcal{S}}$ works, $C'_j(x)$ is a random polynomial under the constraint that $C'_j(0) = f_a(\alpha_j) \cdot f_b(\alpha_j)$. By Corollary 3.3, for any fixed points $\{C'_j(\alpha_i)\}_{i \in I}$ and $C'_j(0)$, it holds that any subset of $t - |I|$ points of $\{C'_j(\alpha_\ell)\}_{\ell \notin I}$ are uniformly distributed (note that none of the points in $\{C'_j(\alpha_\ell)\}_{\ell \notin I}$ are seen by the adversary). This implies that for any $t - |I|$ points α_ℓ (with $\ell \notin I$) the points $\hat{f}_{ab}(\alpha_\ell)$ are uniformly distributed. This is therefore exactly the same as choosing $t - |I|$ values $\beta_\ell \in_R \mathbb{F}$ at random (with $\ell \notin I$), and setting $\hat{f}_{ab}$ to be the unique polynomial such that $\hat{f}_{ab}(\alpha_\ell) = \beta_\ell$ in addition to the above constraints. Thus, $f_{ab}(x)$ and $\hat{f}_{ab}(x)$ are identically distributed, and so the outputs of the honest parties $\{f_{ab}(\alpha_j)\}_{j \notin I}$ and $\{\hat{f}_{ab}(\alpha_j)\}_{j \notin I}$ are identically distributed. ■

7 Secure Computation in the (F_{VSS}, F_{mult})-Hybrid Model

In this section we show how to t-securely compute any functionality f in the (F_{VSS}, F_{mult})-hybrid model, in the presence of a malicious adversary corrupting any $t < n/3$ parties. We also assume that all inputs are in a known field $\mathbb{F}$ (with $|\mathbb{F}| > n$), and that the parties all have an arithmetic circuit C over $\mathbb{F}$ that computes f. As in the semi-honest case, we assume that $f : \mathbb{F}^n \to \mathbb{F}^n$ and so the input and output of each party is a single field element. This means that each party's input is associated with a single circuit-input wire and each party's output is associated with a single circuit-output wire. This is only for the sake of clarity of exposition, and the modifications to the protocol for the general case are straightforward.

The protocol here is almost identical to Protocol 4.1 for the semi-honest case; the only difference is that the verifiable secret-sharing functionality F_{VSS} is used in the input stage, and the F_{mult} functionality used for multiplication gates in the computation stage is the reactive one defined for the case of malicious adversaries. See Section 5.4 for the definition of F_{VSS}, and see Functionality 6.10 in Section 6.4 for the definition of F_{mult}. Observe that the definition of F_{VSS} is such that the effect is identical to that of Shamir secret sharing in the presence of semi-honest adversaries. Furthermore, the correctness of F_{mult} ensures that at every intermediate stage the (honest) parties hold correct shares on the wires of the circuit. In addition, observe that F_{mult} reveals nothing to the adversary except for its points on the input wires, which it already knows. Thus, the adversary learns nothing in the computation stage, and after this stage the parties all hold correct shares on the circuit-output wires. The protocol is therefore concluded by having the parties send their shares on the output wires to the appropriate recipients (i.e., if party P_j is supposed to receive the output on a certain wire, then all parties send their shares on that wire to P_j). This step introduces a difficulty that does not arise in the semi-honest setting; some of the parties may send *incorrect* values on these wires. Nevertheless, this is not a problem since it is guaranteed that more than

$2n/3$ shares are correct and so each party can apply Reed-Solomon decoding to ensure that the final output obtained is correct. See Protocol 7.1 for full details.

PROTOCOL 7.1 (t-Secure Computation in the (F_{mult}, F_{VSS})-Hybrid Model)

- **Inputs:** Each party P_i holds private input $x_i \in \mathbb{F}$
- **Common input:** Each party P_i holds an arithmetic circuit C over the field $\mathbb{F}$, such that for every $\vec{x} \in \mathbb{F}^n$ it holds that $C(\vec{x}) = f(\vec{x})$, where $f : \mathbb{F}^n \to \mathbb{F}^n$. The parties also hold a description of $\mathbb{F}$ and distinct non-zero values $\alpha_1, \ldots, \alpha_n$ in $\mathbb{F}$.
- **The protocol:**
  1. **The input stage:**
     (a) For every $i \in \{1, \ldots, n\}$, party P_i chooses a polynomial $q_i(x)$ uniformly at random from the set $\mathcal{P}^{x_i,t}$. Then, P_i invokes the F_{VSS} functionality as dealer, using $q_i(x)$ as its private input.
     (b) Each party P_i records the values $q_1(\alpha_i), \ldots, q_n(\alpha_i)$ that it received from the F_{VSS} functionality. If the output from F_{VSS} is $\perp$ for any of these values, P_i replaces the value with 0.
  2. **The computation stage:** Let $g_1, \ldots, g_\ell$ be a predetermined topological ordering of the gates of the circuit. For $k = 1, \ldots, \ell$ the parties work as follows:
     - *Case 1 – g_k is an addition gate:* Let β_k^i and γ_k^i be the shares of input wires held by party P_i. Then, P_i defines its share of the output wire to be $\delta_k^i = \beta_k^i + \gamma_k^i$.
     - *Case 2 – g_k is a multiplication-by-a-constant gate with constant c:* Let β_k^i be the share of the input wire held by party P_i. Then, P_i defines its share of the output wire to be $\delta_k^i = c \cdot \beta_k^i$.
     - *Case 3 – g_k is a multiplication gate:* Let β_k^i and γ_k^i be the shares of input wires held by party P_i. Then, P_i sends (β_k^i, γ_k^i) to the ideal functionality F_{mult} and receives back a value δ_k^i. Party P_i defines its share of the output wire to be δ_k^i.
  3. **The output stage:**
     (a) For every circuit-output wire w, the parties work as follows. Let P_j be the party for whom the wire w contains its output, and denote by $\beta_k^1, \ldots, \beta_k^n$ the shares that the parties hold for wire w. Then, each P_i sends P_j its share β_k^i.
     (b) Upon receiving all shares, P_j runs the Reed-Solomon decoding procedure on the shares to obtain a codeword $(\tilde{\beta}_k^1, \ldots, \tilde{\beta}_k^n)$. Then, P_j interpolates in order to obtain the polynomial $q_w(x)$ such that $q_w(\alpha_i) = \tilde{\beta}_k^i$, for every $i \in \{1, \ldots, n\}$. P_j then defines its output to be $q_w(0)$.

Theorem 7.2 *Let $\mathbb{F}$ be a finite field of size greater than n, let $f : \mathbb{F}^n \to \mathbb{F}^n$ be an n-ary functionality, and let $t < n/3$. Then, Protocol 7.1 with auxiliary-input C to all parties t-securely computes f in the (F_{VSS}, F_{mult})-hybrid model, in the presence of a static malicious adversary.*

Proof: Intuitively, security here follows from the fact that a corrupted party in Protocol 7.1 cannot do anything but choose its input as it wishes. In order to see this, observe that the entire protocol is comprised of F_{VSS} and F_{mult} calls, and in the latter the adversary receives no new information in its output and has no influence whatsoever on the outputs of the honest parties. Finally, the adversary cannot affect the outputs of the honest parties due to the Reed-Solomon decoding carried out in the output stage. We now formally describe the simulator.

The Simulator $\mathcal{S}$:

- *$\mathcal{S}$ invokes $\mathcal{A}$ with its auxiliary input z.*
- **The input stage:**
  1. *For every $j \notin I$, $\mathcal{S}$ chooses a uniformly distributed polynomial $q_j(x) \in_R \mathcal{P}^{0,t}$ with free coefficient 0, and simulates F_{VSS} sending each party P_i the value $q_j(\alpha_i)$ for every $i \in I$.*
  2. *For every $i \in I$, $\mathcal{S}$ obtains from $\mathcal{A}$ the polynomial $q_i(x)$ that it instructs P_i to send to the F_{VSS} functionality when P_i is the dealer. $\mathcal{S}$ checks that $\deg(q_i(x)) = t$. If yes, it simulates F_{VSS} sending $q_i(\alpha_\ell)$ to P_ℓ for every $\ell \in I$. If no, it simulates F_{VSS} sending $\perp$ to P_ℓ for every $\ell \in I$, and sets $q_i(x)$ to be constant zero polynomial.*
  3. *For every $k \in \{1, \ldots, n\}$, denote the circuit-input wire that receives P_k's input by w_k. Then, for every $i \in I$, simulator $\mathcal{S}$ stores the value $q_k(\alpha_i)$ as the share of P_i on the wire w_k.*
- **Interaction with the trusted party:**
  1. *$\mathcal{S}$ sends the trusted party computing f the values $\{x_i = q_i(0)\}_{i \in I}$ as the inputs of the corrupted parties.*
  2. *$\mathcal{S}$ receives from the trusted party the outputs $\{y_i\}_{i \in I}$ of the corrupted parties.*
- **The computation stage:** *Let $g_1, \ldots, g_\ell$ be the gates of the circuit. For $k = 1, \ldots, \ell$:*
  1. Case 1 – g_k is an addition gate: *Let β_k^i and γ_k^i be the shares that $\mathcal{S}$ has stored for the input wires to g_k for the party P_i. Then, for every $i \in I$, $\mathcal{S}$ computes the value $\delta_k^i = \beta_k^i + \gamma_k^i$ as the share of P_i for the output wire of g_k and stores this values.*
  2. Case 2 – g_k is a multiplication-by-a-constant gate with constant c: *Let β_k^i be the share that $\mathcal{S}$ has stored for the input wire to g_k for P_i. Then, for every $i \in I$, $\mathcal{S}$ computes the value $\delta_k^i = c \cdot \beta_k^i$ as the share of P_i for the output wire of g_k and stores this value.*
  3. Case 3 – g_k is a multiplication gate: *$\mathcal{S}$ plays the trusted party computing F_{mult} for $\mathcal{A}$, as follows. Let β_k^i and γ_k^i be the shares that $\mathcal{S}$ has stored for the input wires to g_k for the party P_i. Then, $\mathcal{S}$ first hands $\{(\beta_k^i, \gamma_k^i)\}_{i \in I}$ to $\mathcal{A}$ as if coming from F_{mult} (see Step 2 of Functionality 6.10) Next, it obtains from $\mathcal{A}$ values $\{\delta_k^i\}_{i \in I}$ as the input of the corrupted parties for the functionality F_{mult} (if any δ_k^i is not sent, then $\mathcal{S}$ sets $\delta_k^i = 0$). Finally, $\mathcal{S}$ stores δ_k^i as the share of P_i for the output wire of g_k. (Note that the adversary has no output from F_{mult} beyond receiving the (β_k^i, γ_k^i) values.)*
- **The output stage:** *For every $i \in I$, simulator $\mathcal{S}$ works as follows. Denote by w_i' the circuit-output wire that contains the output of party P_i, and let $\{\beta_i^\ell\}_{\ell \in I}$ be the shares that $\mathcal{S}$ has stored for wire w_i' for all corrupted parties P_ℓ ($\ell \in I$). Then, $\mathcal{S}$ chooses a random polynomial $q_i'(x)$ under the constraint that $q_i'(\alpha_\ell) = \beta_i^\ell$ for all $\ell \in I$, and $q_i'(0) = y_i$, where y_i is the output of P_i received by $\mathcal{S}$ from the trusted party computing f. Finally, for every $j \notin I$, $\mathcal{S}$ simulates the honest party P_j sending $q_i'(\alpha_j)$ to P_i.*

An alternative simulator $\mathcal{S}'$: We begin by constructing an alternative simulator $\mathcal{S}'$ that works exactly like $\mathcal{S}$ except that it receives as input all of the input values $\vec{x} = (x_1, \ldots, x_n)$, and chooses the polynomials $q_j(x) \in_R \mathcal{P}^{x_j,t}$ of the honest parties with the correct free coefficient instead of with free coefficient 0. Apart from this, $\mathcal{S}'$ works exactly like $\mathcal{S}$ and interacts with a trusted party computing f in the ideal model.

The original and alternative simulations. We now show that the joint output of the adversary and honest parties is identical in the original and alternative simulations. That is,

$$\left\{\text{IDEAL}_{f,\mathcal{S}(z),I}(\vec{x})\right\}_{\vec{x}\in(\{0,1\}^*)^n, z\in\{0,1\}^*} \equiv \left\{\text{IDEAL}_{f,\mathcal{S}'(\vec{x},z),I}(\vec{x})\right\}_{\vec{x}\in(\{0,1\}^*)^n, z\in\{0,1\}^*}. \quad (16)$$

This follows immediately from the fact that both $\mathcal{S}$ and $\mathcal{S}'$ can work identically when receiving the points $\{q_j(\alpha_i)\}_{i\in I; j\notin I}$ externally. Furthermore, the only difference between them is if $q_j(\alpha_i) \in_R \mathcal{P}^{0,t}$ or $q_j(\alpha_i) \in_R \mathcal{P}^{x_j,t}$, for every $j \notin I$. Thus, there exists a single machine $\mathcal{T}$ that runs in the ideal model with a trusted party computing f, and that receives points $\{q_j(\alpha_i)\}_{i\in I; j\notin I}$ and runs the simulation using these points. Observe that if $q_j(\alpha_i) \in_R \mathcal{P}^{0,t}$ for every $j \notin I$, then the joint output of $\mathcal{T}$ and the honest parties in the ideal execution is exactly the same as in the ideal execution with $\mathcal{S}$. In contrast, if $q_j(\alpha_i) \in_R \mathcal{P}^{x_j,t}$ for every $j \notin I$, then the joint output of $\mathcal{T}$ and the honest parties in the ideal execution is exactly the same as in the ideal execution with the alternative simulator $\mathcal{S}'$. By Claim 3.4, these points are identically distributed in both cases, and thus the joint output of $\mathcal{T}$ and the honest parties are identically distributed in both cases; Eq. (16) follows.

The alternative simulation and a protocol execution. We now proceed to show that:

$$\left\{\text{IDEAL}_{f,\mathcal{S}'(\vec{x},z),I}(\vec{x})\right\}_{\vec{x}\in(\{0,1\}^*)^n, z\in\{0,1\}^*} \equiv \left\{\text{HYBRID}^{F_{VSS},F_{mult}}_{\pi,\mathcal{A}(z),I}(\vec{x})\right\}_{\vec{x}\in(\{0,1\}^*)^n, z\in\{0,1\}^*}.$$

We first claim that the output of the honest parties are identically distributed in the real execution and the alternative simulation. This follows immediately from the fact that the inputs to F_{VSS} fully determine the inputs $\vec{x}$, which in turn fully determine the output of the circuit. In order to see this, observe that F_{mult} always sends shares of the product of the input shares (this holds as long as the honest parties send "correct" inputs which they always do), and the local computation in the case of multiplication-by-a-constant and addition gates is trivially correct. Thus, the honest parties all hold correct shares of the outputs on the circuit-output wires. Finally, by the Reed-Solomon decoding procedure (with code length n and dimension $t+1$), it is possible to correct up to $\frac{n-t}{2} > \frac{3t-t}{2} = t$ errors. Thus, the values sent by the corrupted parties in the output stage have no influence whatsoever on the honest parties' outputs.

Next, we show that the view of the adversary $\mathcal{A}$ in the alternative simulation with $\mathcal{S}'$ is identical to its view in real protocol execution, conditioned on the honest parties' outputs $\{y_j\}_{j\notin I}$. It is immediate that these views are identical up to the output stage. This is because $\mathcal{S}'$ uses the same polynomials as the honest parties in the input stage, and in the computation stage $\mathcal{A}$ receives no output at all (except for its values on the input wires for multiplication gates which are already known). It thus remains to show that the values $\{q'_i(\alpha_j)\}_{i\in I; j\notin I}$ received by $\mathcal{A}$ from $\mathcal{S}'$ in the output stage are identically distributed to the values received by $\mathcal{A}$ from the honest parties P_j. In order to see this, we describe the difference between the way these values are generated in a real protocol execution and in the simulation:

- *Real execution:* Let $\{\beta_i^j\}_{j\notin I}$ be the shares that the honest parties have on the circuit-output wire w_i'; these are the values received by the corrupt P_i in the output stage. Assume for simplicity that these shares are the direct output of a multiplication gate (otherwise, they are a deterministic function of other wires), and let $\{\beta_i^\ell\}_{\ell\in I}$ be the values sent by $\mathcal{A}$ to F_{mult} as input into this multiplication. Then, by the definition of F_{mult}, the shares $\{\beta_i^j\}_{j\notin I}$ are generated by choosing a random polynomial $q_i'(x)$ under the constraint that $q_i'(\alpha_\ell) = \beta_i^\ell$ for every $\ell \in I$, and $q_i'(0) = y_i'$ where y_i' is the output of the circuit C on wire w_i'. Note that this latter constraint regarding the free coefficient holds because the output wire of the multiplication gate is the circuit-output wire w_i'.
- *Simulation:* In the output stage, $\mathcal{S}'$ chooses a random polynomial $q_i'(x)$ under the constraint that $q_i'(\alpha_\ell) = \beta_i^\ell$ for every $\ell \in I$, and $q_i'(0) = y_i$. Then, $\mathcal{S}'$ simulates each P_j sending $\beta_i^j = q_i'(\alpha_j)$ to P_i.

Thus, $\mathcal{S}'$ actually chooses $q_i'(x)$ in *exactly* the same way as F_{mult} in the real execution, and the constraints are identical as long as $y_i' = y_i$. However, this follows from the "correctness" of the computation, in the same way as demonstrated regarding the outputs of the honest parties. This concludes the proof. ■

We conclude with the following corollary that considers the plain model (with private channels). This is obtained by combining Theorems 5.5, 6.2, 6.6, 6.9, 6.12 and 7.2, and using the modular sequential composition theorem of [5]:

Corollary 7.3 *For every functionality $f : \mathbb{F}^n \to \mathbb{F}^n$ and every $t < n/3$, there exists a protocol that t-securely computes f in the plain (private channels) model, in the presence of a static malicious adversary.*

8 Security Under Composition and Adaptive Corruptions

Our proof of the security of the protocol of [4] in the semi-honest and malicious cases relates to the *stand-alone model* and the case of *static corruptions.* In addition, in the information-theoretic setting, we consider perfectly secure (private) channels. In this section, we show that our proof of security for the limited stand-alone model with static corruptions suffices for obtaining security in the much more complex settings of composition and adaptive corruptions. This is made possible due to the fact that the protocol of [4] is *perfectly secure*, and not just statistically secure.

Adaptive security. In general, security in the presence of a static adversary does not imply security in the presence of a malicious adversary, even for perfectly-secure protocols. However, as shown in [7] this does hold for the definition of security of [13]. This definition requires a straight-line black-box simulator, and also the existence of a committal round at which point the transcript of the protocol fully defines all of the parties' inputs. It is not difficult to see that all of the protocols in this paper meet this definition. Applying the result of [7] we can therefore conclude that all of the protocols are secure in the presence of adaptive adversaries under the definition of [13]. Furthermore, any protocol that is secure in presence of adaptive adversaries under the definition of [13] is also secure in the presence of adaptive adversaries under the definition of [5]. We therefore obtain security in the presence of adaptive adversaries "for free". We have the following corollary:

Corollary 8.1 *For every functionality f, there exists a protocol for securely computing f in the presence of adaptive semi-honest adversaries that corrupt up to $t < n/2$ parties, in the private channels model. Furthermore, there exists a protocol for securely computing f in the presence of adaptive malicious adversaries that corrupt up to $t < n/3$ parties, in the private channels model.*

Security under composition. In [20, Theorem 3] it was proven that any protocol that computes a functionality f with perfect security and has a straight-line black-box simulator (as is the case with all of our simulators), securely computes f under the definition of universal composability [6] (or equivalently, concurrent general composition [22]). Furthermore, this holds for both static and adaptive adversaries. Applying this to Corollary 8.1, and using the terminology UC-secure to mean secure under the definition of universal composability, we have the following corollary:

Corollary 8.2 *For every functionality f, there exists a protocol for UC-securely computing f in the presence of adaptive semi-honest adversaries that corrupt up to $t < n/2$ parties, in the private channels model. Furthermore, there exists a protocol for UC-securely computing f in the presence of adaptive malicious adversaries that corrupt up to $t < n/3$ parties, in the private channels model.*

Security in the computational setting. There are two differences between the information-theoretic and computational settings. First, in the information-theoretic setting, the adversary does not necessarily run in polynomial time. Second, in the information-theoretic setting there are ideally private channels, whereas in the computational setting it is typically only assumed that there are authenticated channels. Nevertheless, as advocated by [17, Sec. 7.6.1] the simulators must be polynomial-time in the running-time of the adversary. Thus, if the real adversary runs in polynomial-time, then so does the simulator, as required for the computational setting. Second, it is possible to replace the ideally private channels with public-key encryption for the case of static adversaries and with non-committing encryption [8] for the case of adaptive adversaries. We state our corollary here for the most general setting of adaptive adversaries and UC-security, although analogous corollaries can of course be obtained for the more restricted models as well. This corollary is obtained by replacing the private channels in Corollary 8.2 with UC-secure channels that can be constructed for the static and adaptive setting using semantically-secure public-key encryption and non-committing encryption, respectively [6, 9]. We state the corollary only for the case of malicious adversaries since the case of semi-honest adversaries has already been proven in [10] for any $t < n$.

Corollary 8.3 *Assume the existence of semantically-secure public-key encryption (resp., non-committing encryption). Then, for every functionality f, there exists a protocol for UC-securely computing f in the presence of static (resp., adaptive) malicious adversaries that corrupt up to $t < n/3$ parties, in the authenticated channels model.*

We stress that the above protocol requires no common reference string or other setup (beyond that required for obtaining authenticated channels). This is the first full proof of the existence of such a protocol.

Acknowledgements

We thank Oded Goldreich and Tal Rabin for many helpful discussions.

References

[1] G. Asharov, Y. Lindell and T. Rabin. Perfectly-Secure Multiplication for any $t < n/3$. To appear in *CRYPTO 2011.*

[2] D. Beaver. Multiparty Protocols Tolerating Half Faulty Processors. In *CRYPTO'89*, Springer-Verlag (LNCS 435), pages 560–572, 1990.

[3] D. Beaver. Foundations of Secure Interactive Computing. In *CRYPTO'91*, Springer-Verlag (LNCS 576), pages 377–391, 1991.

[4] M. Ben-Or, S. Goldwasser and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In the 20*th STOC,* pages 1–10, 1988.

[5] R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. In the *Journal of Cryptology*, 13(1):143–202, 2000.

[6] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In the 42*nd FOCS*, pages 136–145, 2001.

[7] R. Canetti, I. Damgård, S. Dziembowski, Y. Ishai and T. Malkin: Adaptive versus Non-Adaptive Security of Multi-Party Protocols. In the *Journal of Cryptology* 17(3):153–207, 2004.

[8] R. Canetti, U. Feige, O. Goldreich and M. Naor. Adaptively Secure Multi-Party Computation. In the 28*th STOC*, pages 639–648, 1996.

[9] R. Canetti and H. Krawczyk. Universally Composable Notions of Key-Exchange and Secure Channels. In *EUROCRYPT 2002*, Springer (LNCS 2332), pages 337–351, 2002.

[10] R. Canetti, Y. Lindell, R. Ostrovsky and A. Sahai. Universally Composable Two-Party and Multi-Party Computation. In the 34*th STOC*, pages 494–503, 2002.

[11] D. Chaum, C. Crépeau and I. Damgård. Multi-party Unconditionally Secure Protocols. In 20*th STOC*, pages 11–19, 1988.

[12] B. Chor, S. Goldwasser, S. Micali and B. Awerbuch. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults. In the 26 *FOCS*, pages 383–395, 1985.

[13] Y. Dodis and S. Micali. Parallel Reducibility for Information-Theoretically Secure Computation. In *CRYPTO 2000*, Springer (LNCS 1880), pages 74–92, 2000.

[14] P. Feldman. Optimal Algorithms for Byzantine Agreement. *PhD thesis, Massachusetts Institute of Technology,* 1988.

[15] P. Feldman and S. Micali. An Optimal Probabilistic Protocol for Synchronous Byzantine Agreement. In the *SIAM Journal on Computing*, 26(4):873-933, 1997.

[16] R. Gennaro, M.O. Rabin and T. Rabin. Simplified VSS and Fact-Track Multiparty Computations with Applications to Threshold Cryptography. In the 17*th PODC*, pages 101–111, 1998.

[17] O. Goldreich. *Foundations of Cryptography: Volume 2 – Basic Applications.* Cambridge University Press, 2004.

[18] O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In 19*th STOC,* pages 218–229, 1987. For details see [17].

[19] S. Goldwasser and L. Levin. Fair Computation of General Functions in Presence of Immoral Majority. In *CRYPTO'90*, Spring-Verlag (LNCS 537), pages 77–93, 1990.

[20] E. Kushilevitz, Y. Lindell and T. Rabin. Information-Theoretically Secure Protocols and Security Under Composition. In the *SIAM Journal on Computing*, 39(5):2090–2112, 2010.

[21] L. Lamport, R. Shostack, and M. Pease. The Byzantine Generals Problem. In the *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[22] Y. Lindell. General Composition and Universal Composability in Secure Multi-Party Computation. In the 44*th FOCS*, pages 394–403, 2003.

[23] R.J. McEliece and D.V. Sarwate. On Sharing Secrets and Reed-Solomon Codes. *Communications of the ACM*, 9(24):583–584, 1981.

[24] S. Micali and P. Rogaway. Secure Computation. Unpublished manuscript, 1992. Preliminary version in *CRYPTO'91*, Springer-Verlag (LNCS 576), pages 392–404, 1991.

[25] M. Pease, R. Shostak, and L. Lamport. Reaching Agreement in the Presence of Faults. In the *Journal of the ACM*, 27(2):228–234, 1980.

[26] T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multi-party Protocols with Honest Majority. In 21*st STOC*, pages 73–85, 1989.

[27] A. Shamir. How to Share a Secret. In the *Communications of the ACM*, 22(11):612–613, 1979.

[28] A. Yao. How to Generate and Exchange Secrets. In 27*th FOCS*, pages 162–167, 1986.

A Communication Complexity

In this section, we summarize the exact communication complexity of the BGW protocol (as presented here) in the case of malicious adversaries. We count the cost in the "optimistic case" where no party deviates from the protocol specification, and the "dishonest case" where some party does. We remark that since the protocol achieves perfect security, nothing can be gained by deviating, except possible to make the parties run longer. Thus, in general, one would expect that the typical cost of running the protocol is the "optimistic cost". In addition, we separately count the number of field elements sent over the point-to-point private channels, and the number of elements sent over a broadcast channel. (The "BGW" row in the table counts the overall cost of computing a circuit C.)

Protocol	**Optimistic Cost**	**Dishonest Cost**
$\boldsymbol{F_{VSS}}$:	$O(n^2)$ over pt-2-pt No broadcast	$O(n^2)$ over pt-2-pt $O(n^2)$ broadcast
$\boldsymbol{F_{VSS}^{subshare}}$:	$O(n^3)$ over pt-2-pt No broadcast	$O(n^3)$ over pt-2-pt $O(n^3)$ broadcast
$\boldsymbol{F_{eval}}$:	$O(n^3)$ over pt-2-pt No broadcast	$O(n^3)$ over pt-2-pt $O(n^3)$ broadcast
$\boldsymbol{F_{VSS}^{mult}}$:	$O(n^3)$ over pt-2-pt No broadcast	$O(n^5)$ over pt-2-pt $O(n^5)$ broadcast
$\boldsymbol{F_{mult}}$:	$O(n^4)$ over pt-2-pt No broadcast	$O(n^6)$ over pt-2-pt $O(n^6)$ broadcast
BGW:	$O(\|C\| \cdot n^4)$ over pt-2-pt No broadcast	$O(\|C\| \cdot n^6))$ over pt-2-pt $O(\|C\| \cdot n^6)$ broadcast