$1/p$-Secure Multiparty Computation without an Honest Majority and the Best of Both Worlds

Amos Beimel*
Department of Computer Science
Ben Gurion University
Be'er Sheva, Israel

Yehuda Lindell†
Department of Computer Science
Bar Ilan University
Ramat Gan, Israel

Eran Omri‡
Department of Computer Science and Mathematics
Ariel University
Ariel, Israel

Ilan Orlov§
Department of Computer Science
Ben Gurion University
Be'er Sheva, Israel

September 10, 2013

Abstract

A protocol for computing a functionality is secure if an adversary in this protocol cannot cause more harm than in an ideal computation, where parties give their inputs to a trusted party that returns the output of the functionality to all parties. In particular, in the ideal model such computation is fair – if the corrupted parties get the output, then the honest parties get the output. Cleve (STOC 1986) proved that, in general, fairness is not possible without an honest majority. To overcome this impossibility, Gordon and Katz (Eurocrypt 2010) suggested a relaxed definition – $1/p$-secure computation – which guarantees partial fairness. For two parties, they constructed $1/p$-secure protocols for functionalities for which the size of either their domain or their range is polynomial (in the security parameter). Gordon and Katz ask whether their results can be extended to multiparty protocols.

We study $1/p$-secure protocols in the multiparty setting for general functionalities. Our main result is constructions of $1/p$-secure protocols that are resilient against *any* number of corrupted parties provided that the number of parties is constant and the size of the range of the functionality is at most polynomial (in the security parameter n). If fewer than 2/3 of the parties are corrupted, the size of the domain of each party is constant, and the functionality is deterministic, then our protocols are efficient even when the number of parties is $\log \log n$. On the negative side, we show that when the number of parties is super-constant, $1/p$-secure protocols are not possible when the size of the domain of each party is polynomial. Thus, our feasibility results for $1/p$-secure computation are essentially tight.

We further motivate our results by constructing protocols with stronger guarantees: If in the execution of the protocol there is a majority of honest parties, then our protocols provide full security. However, if only a minority of the parties are honest, then our protocols are $1/p$-secure. Thus, our protocols provide the best of both worlds, where the $1/p$-security is only a fall-back option if there is no honest majority.

*Generously supported by ISF grant 938/09 and by the Frankel Center for Computer Science.

†Generously supported by the European Research Council as part of the ERC project LAST, and by the ISRAEL SCIENCE FOUNDATION (grant No. 781/07).

‡Generously supported by the European Research Council as part of the ERC project LAST, and by the ISRAEL SCIENCE FOUNDATION (grant No. 781/07).

§Generously supported by ISF grant 938/09 and by the Frankel Center for Computer Science.

Contents

1 Introduction

A protocol for computing a functionality is secure if an adversary in this protocol cannot cause more harm than in an ideal computation, where parties give their inputs to a trusted party, which, in turn, returns the output of the functionality to all parties. This is formalized by requiring that for every adversary in the real world, there is an adversary in the ideal world, called simulator, such that the output of the real-world adversary and the simulator are indistinguishable in polynomial time. Such security can be achieved when there is a majority of honest parties [18]. Secure computation is fair – if the corrupted parties get the output, then the honest parties get the output. Cleve [10] proved that, in general, fairness is not possible without an honest majority.

To overcome the impossibility of [10], Gordon and Katz [24] suggested a relaxed definition – $1/p$-secure computation – which guarantees partial fairness. Informally, a protocol is $1/p$-secure if for every adversary in the real world, there is a simulator running in the ideal world, such that the output of the real-world adversary and the simulator cannot be efficiently distinguished with probability greater than $1/p$. For two parties, Gordon and Katz construct $1/p$-secure protocols for functionalities for whom the size of either their domain or their range is polynomial (in the security parameter). They also give impossibility results when both the domain and range are super-polynomial. Gordon and Katz ask whether their results can be extended to multiparty protocols. We give positive and negative answers to this question.

1.1 Previous Results

Cleve [10] proved that any protocol for coin tossing without an honest majority cannot be fully secure; specifically, if the protocol has r rounds, then it is at most $1/r$-secure. Protocols with partial fairness, under various definitions and assumptions, have been constructed for coin tossing [10, 11, 29, 5], for contract signing/exchanging secrets [7, 28, 13, 6, 12, 8], and for general functionalities [33, 14, 3, 19, 31, 15, 24]. We next describe the papers that are most relevant to our paper. Moran, Naor, and Segev [29] constructed 2-party protocols for coin tossing that are $1/r$-secure (where r is the number of rounds in the protocol). Gordon and Katz [24] defined $1/p$-security and constructed 2-party $1/p$-secure protocols for every functionality whose size of either the domain or the range of the functionality is polynomial. Finally, Beimel, Omri, and Orlov [5] constructed multiparty protocols for coin tossing that are $O(1/r)$-secure provided that the fraction of bad parties is slightly larger than half. In particular, their protocol is $O(1/r)$-secure when the number of parties is constant and the fraction of bad parties is fewer than 2/3.

Gordon et al. [21] showed that complete fairness is possible in the two party case for some functions. Gordon and Katz [23] showed similar results for the multiparty case. The characterization of the functions that can be computed with full fairness without an honest majority is open. Gordon et al. [22] studied completeness for fair computations. Specifically, they showed a specific function that is complete for fair two-party computation; this function is also complete for $1/p$-secure two-party computation.

Ishai et al. [25] considered "best of two worlds" protocols. Such protocols should provide full security with an honest majority and some (weaker) security if there is only a minority of honest parties. They give positive and negative results for the existence of such protocols. We discuss some of their results below.

Recently, Asharov et al. [1] considered the class of deterministic Boolean functions with finite domain, and they asked for which functions in this class is it possible to information-theoretically toss an unbiased coin, given a protocol for securely computing the function with fairness. They provided a complete characterization of the functions in this class that imply and do not imply fair coin tossing. This result is a step to answering the question of which functions cannot be securely computed with fairness.

1.2 Our Results

We study $1/p$-secure protocols in the multiparty setting. We construct protocols for general functionalities that are $1/p$-secure against *any* number of corrupted parties, provided that the number of parties is constant. Our protocols require that the size of the range of the (possibly randomized) functionality is at most polynomial in the security parameter. That is, we show the following feasibility result.

Theorem (Informal)**.** *Let $\mathcal{F}$ be a (possibly randomized) functionality with a constant number of parties for whom the size of range is at most polynomial in the security parameter n. Then, for every polynomial $p(n)$ there is a $1/p(n)$-secure protocol for $\mathcal{F}$ tolerating any number of corrupted parties.*

Our results are the first general feasibility results for $1/p$-secure protocols in the multi-party setting, e.g., even for the case that there are 3 parties and two of them might be corrupted. We provide two additional protocols that are $1/p$-secure assuming that the fraction of corrupted parties is fewer than 2/3. These two protocols are more efficient than the protocols discussed above. Specifically, one of the protocols is $1/p$-secure even when the number of parties is $\log \log n$ (where n is the security parameter) provided that the functionality is deterministic and the size of the domain of inputs is constant.

The definition of $1/p$-security allows that with probability $1/p$ the outputs of the honest parties are arbitrary, e.g., for a Boolean function the outputs can be non-Boolean. Some of our protocols are always correct; that is, they always return an output of the functionality with the inputs of the honest parties and some inputs for the corrupted parties. This correctness property is essential for the best of both worlds results described below.

We further motivate our results by constructing a protocol with best of both worlds guarantees: we construct a single protocol such that (1) If in the execution of the protocol there is a majority of honest parties, then our protocol provides full security. (2) If only a minority of parties is honest, then our protocol is $1/p$-secure. The protocol succeeds although it doed not know in advance if there is an honest majority or not. Specifically, we show that

Theorem (Informal)**.** *Let $\mathcal{F}$ be a functionality with a constant number of parties whose size of domain and range is at most polynomial in the security parameter n. Then, for every polynomial $p(n)$ there is a protocol for $\mathcal{F}$ tolerating any number of corrupted parties such that*

- *If there is an honest majority, then the protocol is fully secure.*
- *If there is no honest majority, then the protocol is $1/p(n)$-secure.*

Thus, the $1/p$-security guarantee can be considered as a fall-back option if there is no honest majority. Our protocol provides the best of both worlds, the world of honest majority where the known protocols (e.g., [18]) provide full security if there is an honest majority and provide no security guarantees if no such majority exists, and the world of secure computation without an honest majority. In the latter world the security is either security-with-abort or $1/p$-security. These types of security are incomparable. Ishai et al. [25] proved that there is no general protocol that provides full security when there is an honest majority and security-with-abort without an honest majority. Thus, our protocol provides the best possible combination of both worlds.

Katz [27] presented a protocol, for any functionality $\mathcal{F}$, with full security when there is an honest majority, as well as $1/p$-security *with abort* for any number of corrupted parties. This result assumes a non-rushing adversary. In contrast, our protocol provides a stronger security with a minority of honest parties and can handle the more realistic case of a rushing adversary. However, our protocol only works with a constant number of parties and a polynomial size domain.

To complete the picture, we prove interesting impossibility results. We show that, in general, when the number of parties is super-constant, $1/p$-secure protocols are not possible without an honest majority when the size of the domain of each party is polynomial. This impossibility result justifies the fact that in our protocols the number of parties is constant. We also show that, in general, when the number of parties is $\omega(\log n)$, $1/p$-secure protocols are not possible without an honest majority even when the size of the domain of each party is 2. The proof of the impossibility results is rather simple and follows from an impossibility result of [24]. Nevertheless, they show that our general feasibility results are almost tight.

Our impossibility results should be contrasted with the coin-tossing protocol of [5] that is an efficient $1/p$-secure protocol even when $m(n)$, the number of parties, is polynomial in the security parameter and the number of bad parties is $m(n)/2 + O(1)$. Our results show that these parameters are not possible for general $1/p$-secure protocols even when the size of the domain of each party is 2.

The above mentioned impossibility results do not rule out that the best of two worlds results of Katz [27] can be strengthened by removing the restriction that the adversary is non-rushing. We show that this is impossible; that is, in general, when the number of parties is super-constant and the size of the domain of each party is polynomial, there is no protocol that is fully secure with an honest majority and $1/p$-secure-with-abort without such a majority.

1.3 The Ideas Behind Our Protocols

Our protocols use ideas from the protocols of Gordon and Katz [24] and Beimel et al. [5], both of which generalize the protocol of Moran, Naor, and Segev [29]. In addition, our protocols introduce new ideas that are required to overcome challenges that do not occur in previous works, e.g., dealing with inputs (in contrast to the scenario of [5]) and dealing with a dishonest majority even after parties abort (in contrast to the scenario of [24]). In particular, in order to achieve resilience against any number of corrupted parties we introduce new techniques for hiding the round in which parties learn the output of an execution. Our protocols proceed in rounds, where in each round's values are given to subsets of parties. There is a special round $i^\star$ in the protocol. Prior to round $i^\star$, the values given to a subset of parties are values that can be computed from the inputs of the parties in this subset; starting from round $i^\star$ the values are the "correct" output of the functionality. The values given to a subset are secret shared such that only if all parties in the subset cooperate can they reconstruct the value. Similar to the protocols of [29, 24, 5], the adversary can cause harm (e.g., bias the output of the functionality) only if it guesses $i^\star$; we show that in our protocols this probability is small and the protocols are $1/p$-secure.

In our protocols that are $1/p$-secure against a fraction of 2/3 corrupted parties (which are described in Section 4), if in some round many (corrupted) parties have aborted and there is a majority of honest parties among the active parties, then the set of active parties reconstructs the value given to this set in the previous round. The mechanism to secret share the values in this protocols is similar to [5]; however, there are important differences in this sharing, as the sharing mechanism of [5] is not appropriate for $1/p$-secure computations of functionalities that depend on inputs. The fact that the protocol proceeds until there is an honest majority imposes some restrictions that imply that the protocol can tolerate only a fraction of 2/3 corrupted parties.

Our protocols that are $1/p$-secure against any number of corrupted parties (which are described in Section 5) take a different route. To describe the ideas of the protocol, we consider only the three-party case, where at most two parties are corrupted. In the protocol if one party aborts, then the remaining two parties execute a two-party protocol for the functionality. Again, this protocol proceeds in rounds, where in each round each party gets a value. If the party in the three-party protocols aborts after round $i^\star$, then all these values are the "correct" output of the functionality. To hide $i^\star$, also prior to $i^\star$, with some probability all

these values must be equal. With the remaining probability, a new $i^\star$ is chosen with uniform distribution for the two-party protocol. In other words, in the two-party protocol prior to the original $i^\star$, with some probability, we chose a "fake" value of 1 for the new $i^\star$ of the two-party protocol.

1.4 Open Problems

In our impossibility results the size of the range of each party is super-polynomial (in the security parameter). However, in all our protocols the size of the range of each party is polynomial. It is open if there is an efficient $1/p$-secure protocol when the number of parties is not constant and the size of both the domain and range of each party is polynomial. In our protocols, the number of rounds is double-exponential in the number of parties. Our impossibility results do not rule out that this double-exponential dependency can be improved.

The protocols of [24] are private – the adversary cannot learn any information on the inputs of the honest parties (other than the information that it can learn in the ideal world of computing $\mathcal{F}$). The adversary can only bias the output. Some of our protocols are provably not private (that is, the adversary can learn extra information). However, for other protocols, we do not know whether they are private. It is open if there are general multiparty $1/p$-secure protocols that are also private.

2 Background and the Model of Computation

A multi-party protocol with m parties is defined by m interactive probabilistic polynomial-time Turing machines $p_1, \ldots, p_m$. Each Turning machine, called party, has the security parameter 1^n as a joint input and a private input y_j. The computation proceeds in rounds. In each round, the active parties broadcast and receive messages on a common broadcast channel. The number of rounds in the protocol is expressed as some function $r(n)$ in the security parameter (typically, $r(n)$ is bounded by a polynomial). At the end of the protocol, the (honest) parties should hold a common value w (which should be equal to an output of a predefined functionality).

In this work we consider a malicious, static, computationally bounded (i.e., non-uniform probabilistic polynomial-time) adversary that controls some subset of parties. That is, before the beginning of the protocol, the adversary corrupts a subset of the parties and may instruct them to deviate from the protocol in an arbitrary way. The adversary has complete access to the internal states of the corrupted parties and fully controls the messages that they broadcast throughout the protocol. The honest parties follow the instructions of the protocol.

The parties communicate via a synchronous network, using only a broadcast channel. The adversary is rushing; that is, in each round the adversary sees the messages broadcast by the honest parties before broadcasting the messages of the corrupted parties for this round (thus, the broadcast messages of the corrupted parties can depend on the messages of the honest parties in the same round).

We consider $1/p$-secure computation. Roughly speaking, we say that a protocol Π is $1/p$-secure if for every adversary $\mathcal{A}$ attacking Π in the real-world there is a simulator $\mathcal{S}$ running in the ideal-world, such that the global output of the real-world and the ideal-world executions cannot be distinguished with probability greater than $1/p$. The formal definitions of $1/p$-security are given in Section 2.2. Security-with-abort and cheat-detection, which is a tool used in this paper, is defined in Section 2.3. The cryptographic tools we use are described in Section 2.5.

2.1 Notations

For an integer ℓ, define $[\ell] = \{1, \ldots, \ell\}$. For a set $L \subseteq [m]$, define $Q_L = \{p_j : j \in L\}$. An m-party functionality $\mathcal{F} = \{f_n\}_{n\in\mathbb{N}}$ is a sequence of polynomial-time computable, randomized mappings $f_n : (X_n)^m \to Z_n$, where $X_n = \{0,1\}^{\ell_d(n)}$ and $Z_n = \{0,1\}^{\ell_r(n)}$ are the domain of inputs of each party and the range respectively; $\ell_d, \ell_r : \mathbb{N} \to \mathbb{N}$ are some fixed functions. We denote the size of the domain and the range of $\mathcal{F}$ by $d(n)$ and $g(n)$, respectively; that is, $d(n) = 2^{\ell_d(n)}$ and $g(n) = 2^{\ell_r(n)}$. For a randomized mapping f_n, the assignment $w \leftarrow f_n(x_1, \ldots, x_m)$ denotes the process of computing f_n with the inputs $x_1, \ldots, x_m$ and with uniformly chosen random coins and assigning the output of the computation to w. If $\mathcal{F}$ is deterministic, we sometimes call it a function. We sometimes omit n from functions of n (for example, we write d instead of $d(n)$).

2.2 The Real vs. Ideal Paradigm

The security of multiparty computation protocols is defined using the real vs. ideal paradigm. In this paradigm, we consider the real-world model, in which protocols are executed. We then formulate an ideal model for executing the task. This ideal model involves a trusted party whose functionality captures the security requirements from the task. Finally, we show that the real-world protocol "emulates" the ideal-world protocol: For any real-life adversary $\mathcal{A}$ there exists an ideal-model adversary $\mathcal{S}$ (called simulator) such that the global output of an execution of the protocol with $\mathcal{A}$ in the real-world model is distributed similarly to the global output of running $\mathcal{S}$ in the ideal model. In both models there are m parties $p_1, \ldots, p_m$ holding a common input 1^n and private inputs $y_1, \ldots, y_m$, respectively, where $y_j \in X_n$ for $1 \leq j \leq m$.

The Real Model. Let Π be an m-party protocol computing $\mathcal{F}$. Let $\mathcal{A}$ be a non-uniform probabilistic polynomial time adversary that gets the input y_j of each corrupted party p_j and the auxiliary input aux. Let $\mathrm{REAL}_{\Pi,\mathcal{A}(\mathrm{aux})}(\vec{y}, 1^n)$, where $\vec{y} = (y_1, \ldots, y_m)$, be the random variable consisting of the view of the adversary (i.e., the inputs of the corrupted parties, its random string, the auxiliary input, and the messages it got) and the output of the honest parties following an execution of Π.

The Ideal Model. The basic ideal model we consider is a model without abort. Specifically, there is an adversary $\mathcal{S}$ which has corrupted a subset B of the parties. The adversary $\mathcal{S}$ has some auxiliary input aux. An ideal execution for the computing $\mathcal{F}$ proceeds as follows:

Send inputs to trusted party: The honest parties send their inputs to the trusted party. The corrupted parties may either send their received input, or send some other input of the same length (i.e., $x_j \in X_n$) to the trusted party, or abort (by sending a special "$\texttt{abort}_j$" message). Denote by $x_1, \ldots, x_m$ the inputs received by the trusted party. If p_j does not send a valid input, then the trusted party selects $x_j \in X_n$ with uniform distribution.[1]

Trusted party sends outputs: The trusted party computes $f_n(x_1, \ldots, x_m)$ with uniformly random coins and sends the output to the parties.

[1] For the simplicity of the presentation of our protocols, we present a slightly different ideal world than the traditional one. In our model there is no default input in the case of an "abort". However, the protocol can be presented in the traditional model, where a predefined default input is used if a party aborts.

Outputs: The honest parties output the value sent by the trusted party, the corrupted parties output nothing, and $\mathcal{S}$ outputs any arbitrary (probabilistic polynomial-time computable) function of its view (its inputs, the output, and the auxiliary input aux).

Let $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S}(\mathrm{aux})}(\vec{y}, 1^n)$ be the random variable consisting of the output of the adversary $\mathcal{S}$ in this ideal world execution and the output of the honest parties in the execution.

2.2.1 $1/p$-Indistinguishability and $1/p$-Secure Computation

As explained in the introduction, some ideal functionalities for computing $\mathcal{F}$ cannot be implemented when there is no honest majority. We use $1/p$-secure computation, defined by [24], to capture the divergence from the ideal worlds.

Definition 2.1 (**$1/p$**-indistinguishability)**.** *A function $\mu(\cdot)$ is* negligible *if for every positive polynomial $q(\cdot)$ and all sufficiently large n it holds that $\mu(n) < 1/q(n)$. A* distribution ensemble $X = \{X_{a,n}\}_{a\in\mathcal{D}_n, n\in\mathbb{N}}$ *is an infinite sequence of random variables indexed by $a \in \mathcal{D}_n$ and $n \in \mathbb{N}$, where $\mathcal{D}_n$ is a domain that might depend on n. For a fixed function $p(n)$, two distribution ensembles $X = \{X_{a,n}\}_{a\in\mathcal{D}_n,n\in\mathbb{N}}$ and $Y = \{Y_{a,n}\}_{a\in\mathcal{D}_n,n\in\mathbb{N}}$ are* computationally $1/p$-indistinguishable*, denoted $X \overset{1/p}{\approx} Y$, if for every non-uniform polynomial-time algorithm D there exists a negligible function $\mu(\cdot)$ such that for every n and every $a \in \mathcal{D}_n$,*

$$\Big|\Pr[D(X_{a,n}) = 1] - \Pr[D(Y_{a,n}) = 1]\Big| \leq \frac{1}{p(n)} + \mu(n).$$

Two distribution ensembles are *computationally indistinguishable*, denoted $X \overset{\mathrm{c}}{\equiv} Y$, if for every $c \in \mathbb{N}$ they are computationally $\frac{1}{n^c}$ -indistinguishable.

We next define the notion of $1/p$-secure computation [24]. The definition uses the standard real/ideal paradigm [17, 9], where we consider a completely fair ideal model (as typically considered in the setting of an honest majority), and require only $1/p$-indistinguishability rather than indistinguishability.

Definition 2.2 (**$1/p$**-secure computation [24])**.** *Let $p = p(n)$ be a function. An $m(n)$-party protocol Π is said to $1/p$-securely compute a functionality $\mathcal{F}$ when there are at most $t(n)$ corrupted parties, if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the real model controlling at most $t(n)$ parties, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ in the ideal model, controlling the same parties as $\mathcal{A}$, such that the following two distribution ensembles are computationally $1/p$-indistinguishable*

$$\left\{\mathrm{IDEAL}_{\mathcal{F},\mathcal{S}(\mathrm{aux})}(\vec{y}, 1^n)\right\}_{\mathrm{aux}\in\{0,1\}^*,\vec{y}\in(X_n)^m,n\in\mathbb{N}} \quad \overset{1/p}{\approx} \quad \left\{\mathrm{REAL}_{\Pi,\mathcal{A}(\mathrm{aux})}(\vec{y}, 1^n)\right\}_{\mathrm{aux}\in\{0,1\}^*,\vec{y}\in(X_n)^m,n\in\mathbb{N}}.$$

We next define statistical distance between two random variables and the notion of perfect $1/p$-secure computation, which implies the notion of $1/p$-secure computation.

Definition 2.3 (statistical distance)**.** *We define the* statistical distance *between two random variables A and B as the function*

$$\mathrm{SD}(A, B) = \frac{1}{2}\sum_{\alpha}\Big|\Pr[A = \alpha] - \Pr[B = \alpha]\Big|.$$

Definition 2.4 (perfect $1/p$-secure computation)**.** *An m-party protocol Π is said to perfectly $1/p$-secure compute a functionality $\mathcal{F}$ with at most $t(n)$ corrupted parties if for every non-uniform adversary $\mathcal{A}$ in the real model controlling at most $t(n)$ parties, there exists a polynomial-time adversary $\mathcal{S}$ in the ideal model controlling the same parties, such that for every $n \in \mathbb{N}$, for every $\vec{y} \in (X_n)^m$, and for every* $\mathrm{aux} \in \{0,1\}^*$

$$\mathrm{SD}\big(\mathrm{IDEAL}_{\mathcal{F},\mathcal{S}(\mathrm{aux})}(\vec{y},1^n), \mathrm{REAL}_{\Pi,\mathcal{A}(\mathrm{aux})}(\vec{y},1^n)\big) \leq \frac{1}{p(n)}.$$

2.3 Security-with-Abort and Cheat-Detection

We next present a definition of secure multiparty computation that is more stringent than standard definitions of secure computation with abort. This definition extends the definition for secure computation as given by Aumann and Lindell [2]. Roughly speaking, the definition requires that one of two events is possible: (1) The protocol terminates normally, and *all* parties receive their outputs, or (2) Corrupted parties deviate from the prescribed protocol; in this case the adversary obtains the outputs of the corrupted parties (but nothing else), and all honest parties are given the identities of some parties that have cheated. The formal definition uses the real vs. ideal paradigm as discussed in Section 2.2. We next describe the appropriate ideal model.

Execution in the ideal model. Let $B \subseteq [m]$ denote the set of indices of corrupted parties controlled by an adversary $\mathcal{A}$. The adversary $\mathcal{A}$ receives an auxiliary input denoted aux. An ideal execution proceeds as follows:

Send inputs to trusted party: The honest parties send their inputs to the trusted party. The corrupted parties may either send their received input, or send some other input of the same length (i.e., $x_j \in X_n$) to the trusted party, or cheat (by sending a special "$\texttt{cheat}_j$" message). Denote by $x_1, \ldots, x_m$ the inputs received by the trusted party. If the trusted party receives a "$\texttt{cheat}_j$" message from at least one party, then, it sends "$\texttt{cheat}_j$" for each cheated party p_j to all honest parties and terminates.

Trusted party sends outputs to adversary: The trusted party computes $w \leftarrow f_n(x_1, \ldots, x_m)$ and sends the output w to the adversary.

Adversary instructs the trusted party to continue or halt: $\mathcal{A}$ sends either a "$\texttt{continue}$" message or $\{\text{"}\texttt{cheat}_j\text{"}\}_{j\in J}$ to the trusted party for some set of indices of corrupted parties $J \subseteq B$. If it sends a "$\texttt{continue}$" message, the trusted party sends w to all honest parties. Otherwise, if the adversary sends $\{\text{"}\texttt{cheat}_j\text{"}\}_{j\in J}$, then the trusted party sends $\{\text{"}\texttt{cheat}_j\text{"}\}_{j\in J}$ to all honest parties.

Outputs: An honest party always outputs the value w it obtained from the trusted party. The corrupted parties output nothing. The adversary $\mathcal{A}$ outputs any (probabilistic polynomial-time computable) function of the auxiliary input aux, the inputs of the corrupted parties, and the value w obtained from the trusted party.

We let $\mathrm{IDEAL}^{\mathrm{CD}}_{\mathcal{F},\mathcal{S}(\mathrm{aux})}(\vec{y},1^n)$ and $\mathrm{REAL}_{\Pi,A(\mathrm{aux})}(\vec{y},1^n)$ be defined as in Section 2.2 (where in this case $\mathrm{IDEAL}^{\mathrm{CD}}_{\mathcal{F},\mathcal{S}(\mathrm{aux})}(\vec{y},1^n)$ refers to the above execution with cheat-detection of $\mathcal{F}$). This ideal model is different from that of [17] in that in the case of a "cheat", the honest parties get output "$\texttt{cheat}_j$" for all the cheated parties and not a $\perp$ symbol. This means that the honest parties *know* at least one identity of the corrupted party that caused a cheat. This cheat-detection is achieved by most multiparty protocols, including that of [18], but not all (e.g., the protocol of [20] does not meet this requirement). Using this notation we define secure computation with abort and cheat-detection.

Definition 2.5 (security-with-abort and cheat-detection)**.** *Let $\mathcal{F}$ and Π be as in Definition 2.2. A protocol Π is said to* `securely compute` *$\mathcal{F}$ against at most $t(n)$ corrupted parties with abort and cheat-detection if for every non-uniform polynomial-time adversary $\mathcal{A}$ in the real model controlling at most $t(n)$ parties, there exists a non-uniform polynomial-time adversary $\mathcal{S}$ in the ideal model controlling the same parties, such that*

$$\left\{\mathrm{IDEAL}^{\mathrm{CD}}_{\mathcal{F},\mathcal{S}(\mathrm{aux})}(\vec{y},1^n)\right\}_{\mathrm{aux}\in\{0,1\}^*,\vec{y}\in(X_n)^m,n\in\mathbb{N}} \quad \stackrel{c}{\equiv} \quad \left\{\mathrm{REAL}_{\Pi,A(\mathrm{aux})}(\vec{y},1^n)\right\}_{\mathrm{aux}\in\{0,1\}^*,\vec{y}\in(X_n)^m,n\in\mathbb{N}}.$$

For the simplicity of presentation, we assume that the adversarial behavior is restricted only to premature aborting. Therefore, in the rest of the paper, we replace the "$\texttt{cheat}_j$" message with the "$\texttt{abort}_j$" message. We achieve this restriction by using several cryptographic tools such as signatures.

2.4 A Useful Lemma

For analyzing our protocols we need an extension of Lemma 2 from [24]. The lemma considers an abstract game Γ between a challenger and an (unbounded) adversary $\mathcal{A}$. The game is parameterized by values α, β and a number of round r. Let D_1 and D_2 be two arbitrary distributions such that the following inequality holds:

$$\Pr_{z\leftarrow D_2}\left[\Pr_{a\leftarrow D_1}[a=z] \geq \alpha \Pr_{a\leftarrow D_2}[a=z]\right] \geq 1-\beta. \tag{1}$$

We next define the game $\Gamma(r)$ as follows:

1. The challenger chooses $i^\star$ uniformly at random from $[r]$ and then chooses $a_1,\ldots,a_r$ as follows:
   - For $1 \leq i < i^\star$, it chooses $a_i \leftarrow D_1$.
   - For $i^\star \leq i \leq r$, it chooses $a_i \leftarrow D_2$.
2. The challenger and the adversary $\mathcal{A}$ interact in a sequence of at most r rounds. In round i:
   - The challenger gives a_i to the adversary.
   - The adversary responds by an `abort` instruction that stops the game or by a `continue` instruction that make the game proceed to the next round.
3. $\mathcal{A}$ is declared the winner in this game if it aborts in round $i^\star$.

Let $\mathrm{win}(r)$ denote the maximum probability for $\mathcal{A}$ to win this type of game.

Lemma 2.6. *For any two distributions D_1 and D_2 satisfying Inequality (1), it holds that* $\mathrm{win}(r) \leq 1/(\alpha r)+\beta$.

In Lemma 2 of [24], $\beta = 0$. The proof of Lemma 2.6 is similar to the original proof of Lemma 2 of [24]. For completeness, we provider the proof in Appendix A.

Jumping a head, our protocols in this paper have a special $i^\star$-round and it is essential that the adversary cannot guess its value with a high probability, thus, we first prove an lower-bound on the value of α and then, we use Lemma 2.6 to upper-bound the probability of guessing the value of $i^\star$ in the resulting game.

2.5 Cryptographic Tools

We next informally describe two cryptographic tools and one standard cryptographic assumption that appear in our protocols. Formal definitions can be found in, e.g., [16, 17].

Signature Schemes. Informally, a signature on a message proves that the message was created by its presumed sender, and its content was not altered. A signature scheme is a triple $(\mathrm{Gen}, \mathrm{Sign}, \mathrm{Ver})$ containing the key generation algorithm Gen, which outputs a pair of keys, the signing key K_S and the verification key K_v, the signing algorithm Sign, and the verifying algorithm Ver. We assume that it is infeasible to produce signatures without holding the signing key. For formal definition see [17].

Secret-Sharing Schemes. An α-out-of-m secret-sharing scheme is a mechanism for sharing data among a set of parties such that every set of size α can reconstruct the secret, while any smaller set knows nothing about the secret. In this paper, we use two schemes: the XOR-based m-out-of-m scheme (i.e., in this scheme $\alpha = m$) and Shamir's α-out-of-m secret-sharing scheme [32] which is used when $\alpha < m$. In both schemes, for every $\alpha - 1$ parties, the shares of these parties are uniformly distributed and independent of the secret. Furthermore, given such $\alpha - 1$ shares and a secret s, one can *efficiently* complete them to m shares of the secret s.

In our protocols we sometimes require that in reconstruction of secret only a single party learns the value of a secret that is shared among all parties. Since all messages are sent over a broadcast channel, we use two layers of secret sharing to obtain the above requirements as described below.

Construction 2.7 (secret sharing with respect to a certain party)**.** *Let s be a secret taken from some finite field $\mathbb{F}$. We share s among m parties* with respect to *a (special) party p_j in an α-out-of-m secret-sharing scheme as follows:*

1. *Choose shares $(s^{(1)}, s^{(2)})$ of the secret s in a two-out-of-two secret-sharing scheme (that is, select $s^{(1)} \in \mathbb{F}$ uniformly at random and compute $s^{(2)} = s - s^{(1)}$). Denote these shares by $\mathrm{mask}_j(s)$ and $\mathrm{comp}(s)$, respectively.*
2. *Compute shares $(\lambda^{(1)}, \ldots, \lambda^{(j-1)}, \lambda^{(j+1)}, \ldots, \lambda^{(m)})$ of the secret $\mathrm{comp}(s)$ in an $(\alpha-1)$-out-of-$(m-1)$ Shamir's secret-sharing scheme. For each $\ell \neq j$, denote $\mathrm{comp}_\ell(s) = \lambda^{(\ell)}$.*

Output:

- *The share of party p_j is $\mathrm{mask}_j(s)$. We call this share "p_j's* masking *share".*
- *The share of each party p_ℓ, where $\ell \neq j$, is $\mathrm{comp}_\ell(s)$. We call this share "p_ℓ's* complement *share".*

In the above scheme, we share the secret s among the parties in P such that only sets of size α that contain p_j can reconstruct the secret. In this construction, for every $\beta < \alpha$ parties, the shares of these parties are uniformly distributed and independent of the secret. Furthermore, given such $\beta < \alpha$ shares and a secret s, one can *efficiently* complete them to m shares of the secret s. In addition, given β shares and a secret s, one can *efficiently* select uniformly at random a vector of shares competing the β shares to m shares of s.

Trapdoor Permutations. A trapdoor permutation is a collection of permutations that (1) are easy to compute, (2) hard to invert in polynomial time, and, (3) are easy to invert given some additional key. The existence of trapdoor permutations is a standard cryptographic assumption and it is used in achieving many cryptographic constructions, e.g., public key encryption and oblivious transfer.

For completeness, we next give an informal definition of trapdoor (one way) permutations. A collection of trapdoor permutations is a set $F = \{f_i : D_i \to D_i\}_{i \in I}$ where I is a set of finite indices and for every $i \in I$, f_i is a permutation such that (1) (Easy to sample function) There exists a probabilistic polynomial time (abbreviated as PPT) algorithm Gen which on input 1^n outputs (i, t_i) where t_i is the trapdoor of i, (2) (Easy to sample domain) There exists a PPT algorithm which on input $i \in I$ outputs $x \in D_i$, (3) (Easy to evaluate function) There exists a PPT algorithm A which on inputs $i \in I$ and $x \in D_i$, computes $\mathrm{A}(i, x) = f_i(x)$, (4) (Hard to invert) Every PPT algorithm fails to invert $y = f_i(x), i$ where $x \in D_i$ with a noticeable probability, and (5) (Easy to invert when trapdoor is known) There exists a PPT algorithm B such that $\mathrm{B}(i, t_i, f_i(x)) = x$.

3 Feasibility Results for $1/p$-Secure Multiparty Computation

In this section we state our main feasibility results. Our main result asserts that any functionality with a polynomial size range for a constant number of parties can be $1/p$-securely computed in polynomial time tolerating any number of corrupted (malicious) parties. We next formally state this result.

Theorem 1. *Let $\mathcal{F}$ be a randomized functionality where m is constant. If enhanced trap-door permutations exist and $d(n)$ and $g(n)$ (the domain and range size, respectively) are bounded by polynomials, then for any polynomial $p(n)$ there is an $r(n)$-round $1/p(n)$-secure protocol computing $\mathcal{F}$ tolerating up to $m(n) - 1$ corrupted parties, where $r(n) = p(n)^2 \cdot (g(n) \cdot d(n)^m)^{O(2^m)}$.*
If $\mathcal{F}$ is deterministic and $d(n)$ is bounded by a polynomial, then there is an $r(n)$-round $1/p(n)$-secure protocol for $r(n) = p(n)^2 \cdot d(n)^{O(m \cdot 2^m)}$.

Theorem 2. *Let $\mathcal{F}$ be an m-party (possibly randomized) functionality. If enhanced trap-door permutations exist, and if m is constant and the size of the range $g(n)$ is bounded by a polynomial in the security parameter n, then for any polynomial $p(n)$ there is an $r(n)$-round $1/p(n)$-secure protocol computing $\mathcal{F}$ tolerating up to $m - 1$ corrupted parties, where $r(n) = \Big(p(n) \cdot g(n)\Big)^{2^{O(m)}}$.*

We give substantially better protocols secure against an adversary that may corrupt strictly fewer than two-thirds of the parties. Formally, we prove the following theorem.

Theorem 3. *Let $\mathcal{F}$ be a (possibly randomized) functionality and $t \in \mathbb{N}$ be such that $m/2 \leq t < 2m/3$. Let $r(n) = p(n) \cdot (2 \cdot d(n)^m \cdot g(n) \cdot p(n))^{2^t}$. If $r(n)$ is bounded by a polynomial in n and enhanced trap-door permutations exist, then there is an $r(n)$-round $1/p(n)$-secure protocol for computing $\mathcal{F}$, tolerating up to t corrupted parties. Furthermore, if $\mathcal{F}$ is deterministic, we let $r(n) = p(n) \cdot d(n)^{m \cdot 2^t}$, and obtain an $r(n)$-round protocol for computing $\mathcal{F}$, with the same properties and under the same assumptions.*

Theorem 4. *Let $\mathcal{F}$ be an $m(n)$-party (possibly randomized) functionality. Let $t(n)$ be such that $m(n)/2 \leq t(n) < 2m(n)/3$. If enhanced trap-door permutations exist, then for any polynomial $p(n)$ the following hold:*

- *If $m(n)$ is constant (hence, $t = t(n)$ is constant) and the size of the range $g(n)$ is bounded by a polynomial, then there exists an $r(n)$-round $1/p(n)$-secure protocol computing $\mathcal{F}$ tolerating up to t corrupted parties, where $r(n) = (2p(n))^{2^t+1} \cdot g(n)^{2^t}$.*

- *If $\mathcal{F}$ is deterministic and the size of the domain $d(n)$ is bounded by a polynomial, then there exists an $r(n)$-round $1/p(n)$-secure protocol computing $\mathcal{F}$ tolerating up to $t(n)$ corrupted parties, where $r(n) = p(n) \cdot d(n)^{m(n) \cdot 2^{t(n)}}$, provided that $r(n)$ is bounded by a polynomial.*

The protocols that imply the results of Theorem 4 are presented in Section 4. As implied by the second item of Theorem 4, the round complexity of our protocol when $\mathcal{F}$ is deterministic has only a linear dependency on $p(n)$. Specifically, this protocol has polynomially many rounds even when the number of parties is $0.5 \log \log n$ provided that the functionality is deterministic and the size of the domain of inputs is constant.

4 Protocols with Less Than Two-Thirds Corrupted Parties

In this section we describe our protocols that are secure when the adversary corrupts strictly fewer than two-thirds of the parties. We start with a protocol that assumes that either the functionality is deterministic and the size of the domain is polynomial, or that the functionality is randomized and both the domain and range of the functionality are polynomial. We then present a modification of the protocol that is $1/p$-secure for (possibly randomized) functionalities if the size of the range is polynomial (even if the size of the domain of $\mathcal{F}$ is not polynomial). The first protocol is more efficient for deterministic functionalities with polynomial-size domain. Furthermore, the first protocol has full correctness, while in the modified protocol, correctness is only guaranteed with probability $1 - 1/p$.

Following [29, 5], we present the first protocol in two stages. We first describe in Section 4.1 a protocol with a dealer and then in Section 4.2 present a protocol without this dealer. The goal of presenting the protocol in two stages is to simplify the understanding of the protocol and to enable us to prove the protocol in a modular way. In Section 4.3, we present a modification of the protocol that is $1/p$-secure if the size of the range is polynomial (even if the size of the domain of f is not polynomial).

4.1 The Protocol for Polynomial-Size Domain with a Dealer

In this section we assume that there is a special trusted on-line dealer, denoted T. This dealer interacts with the parties in rounds, sending messages on private channels. We assume that the dealer knows the set of corrupted parties. In Section 4.2, we show how to remove this dealer and construct a protocol without a dealer.

In our protocol the dealer sends in each round values to subsets of parties; the protocol proceeds with the normal execution as long as at least $t + 1$ of the parties are still active. If in some round i, there are at most t active parties, then the active parties reconstruct the value given to them in round $i - 1$, output this value, and halt. Following [27, 21, 29, 24, 5], the dealer chooses at random with uniform distribution a special round $i^\star$. Prior to this round the adversary gets no information and if the corrupted parties abort the execution prior to $i^\star$, then they cannot bias the output of the honest parties or cause any harm. After round $i^\star$, the output of the protocol is fixed, and also in this case the adversary cannot affect the output of the honest parties. The adversary can cause harm only if it guesses $i^\star$ and this happens with small probability.

We next give a verbal description of the protocol. This protocol is designed such that the dealer can be removed from it in Section 4.2. A formal description of the protocol is given in Figure 1.

At the beginning of the protocol each party sends its input y_j to the dealer. The corrupted parties may send any values of their choice. Let $x_1, \ldots, x_m$ denote the inputs received by the dealer. If a corrupted party p_j does not send an input, then the dealer sets x_j to be a random value selected uniformly from the input domain X_n. In a preprocessing phase, the dealer T selects uniformly at random a special round $i^\star \in \{1, \ldots, r\}$. The dealer computes $w \leftarrow f_n(x_1, \ldots, x_m)$. Then, for every round $1 \leq i \leq r$ and every

Inputs: Each party p_j holds a private input $y_j \in X_n$ and the joint input: the security parameter 1^n, the number of rounds $r = r(n)$, and a bound $t = t(n)$ on the number of corrupted parties.

Instructions for each honest party p_j: (1) After receiving the "`start`" message, send input y_j to the dealer. (2) If the premature termination step is executed with $i = 1$, then send its input y_j to the dealer. (3) Upon receiving output z from the dealer, output z. (Honest parties do not send any other messages throughout the protocol.)

Instructions for the (trusted) dealer:

The preprocessing phase:

1. Set $D_0 = \emptyset$ and send a "`start`" message to all parties.
2. Receive an input, denoted x_j, from each party p_j. For every p_j that sends an "$\texttt{abort}_j$" message, notify all parties that party p_j aborted, select $x_j \in X_n$ with uniform distribution, and update $D_0 = D_0 \cup \{j\}$.
3. Let $D = D_0$. If $|D| \geq m - t$, go to premature termination with $i = 1$.
4. Set $w \leftarrow f_n(x_1, \ldots, x_m)$ and select $i^\star \in \{1, \ldots, r\}$ with uniform distribution.
5. For each $1 \leq i < i^\star$, for each $L \subseteq [m] \setminus D_0$ s.t. $m - t \leq |L| \leq t$: for each $j \in L$ set $\widehat{x}_j = x_j$, for each $j \notin L$ select uniformly at random $\widehat{x}_j \in X_n$, and set $\sigma_L^i \leftarrow f_n(\widehat{x}_1, \ldots, \widehat{x}_m)$.
6. For each $i^\star \leq i \leq r$ and for each $L \subseteq [m] \setminus D_0$ s.t. $m - t \leq |L| \leq t$, set $\sigma_L^i = w$.
7. Send "`proceed`" to all parties.

Interaction rounds: In each round $1 \leq i \leq r$, interact with the parties in three phases:

- **The peeking phase:** For each $L \subseteq [m] \setminus D_0$ s.t. $m - t \leq |L| \leq t$, if Q_L contains only corrupted parties, send the value σ_L^i to all parties in Q_L.
- **The abort phase:** Upon receiving an "$\texttt{abort}_j$" message from a party p_j, notify all parties that party p_j aborted (ignore all other types of messages) and update $D = D \cup \{j\}$. If $|D| \geq m - t$, go to premature termination step.
- **The main phase:** Send "`proceed`" to all parties.

Premature termination step:

- If $i = 1$, then: Receive an input, denoted x'_j, from each active party p_j. For every party p_j that sends an "$\texttt{abort}_j$" message, update $D = D \cup \{j\}$ and select $x'_j \in X_n$ with uniform distribution. Set $w' \leftarrow f_n(x'_1, \ldots, x'_m)$.
- Else, if $i > 1$, then: For each "$\texttt{abort}_j$" message received from a party p_j, update $D = D \cup \{j\}$. Set $w' = \sigma_L^{i-1}$ for $L = [m] \setminus D$.
- Send w' to each party p_j s.t. $j \notin D_0$ and halt.

Normal termination: If the last round of the protocol is completed, send w to to each party p_j s.t. $j \notin D_0$

.

Figure 1: Protocol $\mathrm{MPCWithDLtd}_r$.

$L \subset \{1, \ldots, m\}$ such that $m - t \leq |L| \leq t$, the dealer selects an output, denoted σ_L^i, as follows (this output is returned by the parties in $Q_L = \{p_j : j \in L\}$ if the protocol terminates in round $i+1$ and Q_L is the set of the active parties):

CASE I: $1 \leq i < i^\star$. For every $j \in L$ the dealer sets $\widehat{x}_j = x_j$ and for every $j \notin L$ it chooses $\widehat{x}_j$ independently with uniform distribution from the domain X_n; it computes the output $\sigma_L^i \leftarrow f_n(\widehat{x}_1, \ldots, \widehat{x}_m)$.

CASE II: $i^\star \leq i \leq r$. The dealer sets $\sigma_L^i = w$.

The dealer T interacts with the parties in rounds, where in round i, for $1 \leq i \leq r$, there are of three phases:

The peeking phase. The dealer T sends to the adversary all the values σ_L^i such that all parties in Q_L are corrupted.

The abort and premature termination phase. The adversary sends to T the identities of the parties that abort in the current round. If there are fewer than $t+1$ active parties, then T sends σ_L^{i-1} to the active parties, where Q_L is the set of the active parties; the parties can also abort during this phase (see exact details in Figure 1). The honest parties return this output and halt.

The main phase. If at least $t+1$ parties are active, T notifies the active parties that the protocol proceeds normally to the next round.

If after r rounds there are at least $t+1$ active parties, then T sends w to all active parties and the honest parties output this value.

Example 4.1. *As an example, assume that $m = 5$ and $t = 3$. In this case the dealer computes a value σ_L^i for every set of size 2 or 3. Consider an execution of the protocol where p_1 aborts in round 4 and p_3 and p_4 abort in round 100. In this case, T sends $\sigma_{\{2,5\}}^{99}$ to p_2 and p_5, which return this output.*

The formal proof of the $1/p$-security of the protocol appears in Section 4.4. We next hint why for deterministic functionalities, an adversary can cause harm in the above protocol by at most $O(d^{O(1)}/r)$, where $d = d(n)$ is the size of the domain of the inputs and the number of parties, i.e., m, is constant. As in the protocols of [29, 24, 5], the adversary can only cause harm by causing the protocol to terminate in round $i^\star$. In our protocol, if in some round there are two values σ_L^i and $\sigma_{L'}^i$ that the adversary can obtain such that $\sigma_L^i \neq \sigma_{L'}^i$, then the adversary can deduce that $i < i^\star$. Furthermore, the adversary might have some auxiliary information on the inputs of the honest parties; thus, the adversary might be able to deduce that a round is not $i^\star$ even if all the values that it gets are equal. However, there are fewer than 2^t values that the adversary can obtain in each round (i.e., the values of subsets of the t corrupted parties of size at least $m - t$). We will show that for a round i such that $i < i^\star$, the probability that all these values are equal to a fixed value is $1/d^{O(1)}$ for a deterministic function f_n (for a randomized functionality this probability also depends on the size of the range). By Lemma 2.6, this implies that the protocol is $d^{O(1)}/r$-secure.

4.2 Eliminating the Dealer of the Protocol

We eliminate the trusted on-line dealer in a few steps using a few layers of secret-sharing schemes. First, we change the on-line dealer, so that, in each round i, it shares the value σ_L^i of each subset Q_L among the parties of Q_L using a $|L|$-out-of-$|L|$ secret-sharing scheme – called *inner* secret-sharing scheme. As in Protocol $\mathrm{MPCWithDLtd}_r$ described in Figure 1, the adversary is able to obtain information on σ_L^i only if it controls

all the parties in Q_L. On the other hand, the honest parties can reconstruct σ_L^{i-1} (without the dealer), where Q_L is the set of active parties containing the honest parties. In the reconstruction, if an active (corrupted) party does not give its share, then it is removed from the set of active parties Q_L. This is possible since in the case of a premature termination an honest majority among the active parties is guaranteed (as further explained below).

Next, we convert the on-line dealer to an off-line dealer. That is, we construct a protocol in which the dealer sends only one message to each party in an initialization stage; the parties interact in rounds using a broadcast channel (without the dealer) and in each round i each party learns its shares of the ith round inner secret-sharing schemes. In each round i, each party p_j learns a share of σ_L^i in a $|L|$-out-of-$|L|$ secret-sharing scheme, for every set Q_L such that $j \in L$ and $m - t \leq |L| \leq t$ (that is, it learns its share of the inner scheme). For this purpose, the dealer computes, in a preprocessing phase, the appropriate shares for the inner secret-sharing scheme. For each round, the shares of each party p_j are then shared in a 2-out-of-2 secret-sharing scheme, where p_j gets one of the two shares (this share is a mask, enabling p_j to privately reconstruct its shares of the appropriate σ_L^i although messages are sent on a broadcast channel). All other parties get shares in a t-out-of-$(m-1)$ Shamir secret-sharing scheme of the other share of the 2-out-of-2 secret sharing. See Construction 2.7 for a formal description. We call the resulting secret-sharing scheme the *outer* $(t+1)$-out-of-m scheme (since t parties and the holder of the mask are needed to reconstruct the secret).

To prevent corrupted parties from cheating, by say, sending false shares and causing reconstruction of wrong secrets, every message that a party should send during the execution of the protocol is signed in the preprocessing phase (together with the appropriate round number and with the party's index). In addition, the dealer sends a verification key to each of the parties. To conclude, the off-line dealer gives each party the signed shares for the outer secret-sharing scheme together with the verification key.

A formal description of the functionality of the off-line dealer, called Functionality $\mathrm{ShareGenLtd}$, is given in Figure 2.

The protocol with the off-line dealer proceeds in rounds. In round i of the protocol, all parties broadcast their (signed) shares in the outer $(t+1)$-out-of-m secret-sharing scheme. Thereafter, each party can unmask the message it receives (with its share in the appropriate 2-out-of-2 secret-sharing scheme) to obtain its shares in the $|L|$-out-of-$|L|$ inner secret sharing of the values σ_L^i (for the appropriate sets Q_L's to which the party belongs). If a party stops broadcasting messages or broadcasts improperly signed messages, then all other parties consider it as aborted. If $m - t$ or more parties abort, the remaining parties reconstruct the value of the set that contains all of them, i.e., σ_L^{i-1}. If the premature termination occurs in the first round, then the remaining active parties engage in a fully secure protocol (with honest majority) to compute f_n.

The use of the outer secret-sharing scheme with threshold $t+1$ plays a crucial role in eliminating the on-line dealer. On the one hand, it guarantees that an adversary, corrupting at most t parties, cannot reconstruct the shares of round i before round i. On the other hand, at least $m - t$ parties must abort to prevent the reconstruction of the outer secret-sharing scheme (this is why we cannot proceed after $m-t$ parties aborted). Furthermore, since $t \leq 2m/3$, when at least $m - t$ corrupted parties aborted, there is an honest majority. To see this, assume that at least $m - t$ corrupted parties aborted. Thus, at most $t-(m-t) = 2t-m$ corrupted parties are active. There are $m - t$ honest parties (which are obviously active); therefore, as $2t-m < m - t$ (since $t < 2m/3$), an honest majority is achieved when at least $m - t$ parties abort. In this case we can execute a protocol with full security for the reconstruction.

Finally, we replace the off-line dealer by using a secure-with-abort and cheat-detection protocol computing the functionality computed by the dealer; that is, Functionality $\mathrm{ShareGenLtd}_r$. This is done similarly to the preprocessing phase in [5], which in turn uses the results of [30, 4]. Obtaining the outputs of this

computation, an adversary is unable to infer any information regarding the input of honest parties or the output of the protocol (since it gets t shares of a $(t+1)$-out-of-m secret-sharing scheme). The adversary, however, can prevent the execution, at the price of at least one corrupted party being detected as a cheater by all other parties. In such an event, the remaining parties will start over without the detected cheating party. This goes on either until the protocol succeeds or there is an honest majority and a fully secure protocol computing f_n is executed.

A formal description of the protocol appears in Figure 3. The reconstruction functionality used in this protocol (when at least $m-t$ parties aborted) appears in Figure 4. The details of how to construct a protocol secure-with-abort and cheat-detection with $O(1)$ rounds are given in [5].

Comparison with the multiparty coin-tossing protocol of [5]. Our protocol combines ideas from the protocols of [24, 5]. However, there are some important differences between our protocol and the protocol of [5]. In the coin-tossing protocol of [5], the bits σ_L^i are shared using a threshold scheme where the threshold is smaller than the size of the set Q_L. This means that a proper subset of Q_L containing corrupted parties can reconstruct σ_L^i. In coin tossing this is not a problem since there are no inputs. However, when computing functionalities with inputs, such σ_L^i might reveal information on the inputs of honest parties in Q_L, and we share σ_L^i with threshold $|Q_L|$. As a result, we use more sets Q_L than in [5] and the bias of the protocol is increased (put differently, to keep the same security, we need to increase the number of rounds in the protocol). For example, the protocol of [5] has small bias when there are polynomially many parties and $t = m/2$. Our protocol is efficient only when there is a constant number of parties. As explained in Section 7.1, this difference is inherent as a protocol for general functionalities with polynomially many parties and $t = m/2$ cannot have a small bias.

4.3 A $1/p$-Secure Protocol for Polynomial Range

Using an idea of [24], we modify our protocol so that it will have a small bias when the size of the range of the functionality $\mathcal{F}$ is polynomially bounded (even if $\mathcal{F}$ is randomized and has a big domain of inputs). The only modification is the way that each σ_L^i is chosen prior to round $i^\star$: with probability $1/(2p)$ we choose σ_L^i as a random value in the range of f_n and with probability $1-1/(2p)$ we choose it as in Figure 2. Formally, in the protocol with the dealer, in the preprocessing phase of $\mathrm{MPCWithDLtd}_r$ described in Figure 1, we replace Step (5) with the following step:

- For each $i \in \{1,\ldots,i^\star-1\}$ and for each $L \subseteq [m] \setminus D_0$ s.t. $m-t \leq |L| \leq t$,
  - with probability $1/(2p)$, select uniformly at random $z_L^i \in Z_n$ and set $\sigma_L^i = z_L^i$.
  - with the remaining probability $1-1/(2p)$,
    1. For every $j \notin L$ select uniformly at random $\widehat{x}_j \in X_n$ and for each $j \in L$, set $\widehat{x}_j = x_j$.
    2. Compute $\sigma_L^i \leftarrow f_n(\widehat{x}_1,\ldots,\widehat{x}_m)$.

Similarly, in the protocol without the dealer, Protocol MPCLtd_r, we replace Step (3) in $\mathrm{ShareGenLtd}_r$ (described in Figure 2) with the above step. Denote the resulting protocols with and without the dealer models by $\mathrm{MPCWDPolyRangeLtd}$ and $\mathrm{MPCPolyRangeLtd}_r$, respectively.

The idea why this change improves the protocol is that now the probability that all values held by the adversary are equal prior to round $i^\star$ is larger, thus, the probability that the adversary guesses $i^\star$ is smaller. This modification, however, can cause the honest parties to output a value that is not possible given their inputs, and, in general, we cannot simulate the case (which happens with probability $1/(2p)$) when the output is chosen with uniform distribution from the range.

4.4 Proof of $1/p$-Security of the Protocols with a Dealer for Less Than Two-Thirds Corrupted Parties

In this section we prove that our protocol described in Figure 2, which assumes an trusted dealer, is a perfect $1/p$-secure implementation of the ideal functionality $\mathcal{F}$. We start by presenting in Section 4.4.1 a simulator for Protocol $\mathrm{MPCWithDLtd}_r$. In Section 4.4.2, we prove the correctness of the simulation by showing the global output in the ideal-world is distributed within $1/p$ statistical distance from the global output in the real-world. In Section 4.4.3, we describe the required modifications to the simulator for the protocol for $\mathcal{F}$ that has a polynomial-size range, and argue that the modified simulation is correct.

4.4.1 The Simulator for Protocol $\mathbf{MPCWithDLtd}_r$

We next present a simulator $\mathcal{S}_T$ for Protocol $\mathrm{MPCWithDLtd}_r$, described in Figure 1. Let B be the set of indices of corrupted parties in the execution.

The simulator $\mathcal{S}_T$ invokes $\mathcal{A}$ on the set of inputs $\{y_j : j \in B\}$, the security parameter 1^n, and the auxiliary input aux playing the role of the trusted dealer in the interaction with $\mathcal{A}$.

Simulating the preprocessing phase:

1. $D_0 = \emptyset$.
2. The simulator $\mathcal{S}_T$ sends a "`start`" message to all corrupted parties.
3. $\mathcal{S}_T$ receives a set of inputs $\{x_j : j \in B\}$ that $\mathcal{A}$ submits to the computation of the dealer. If $\mathcal{A}$ does not submit an input on behalf of at least one party, i.e., $\mathcal{A}$ sends an "$\texttt{abort}_j$" message for at least one party p_j, then, for each aborted party p_j, the simulator $\mathcal{S}_T$ notifies all corrupted parties that p_j aborted and updates $D_0 = D_0 \cup \{j\}$.
4. $\mathcal{S}_T$ sets $D = D_0$. If $|D| \geq m - t$, the simulator sets $i = 1$ and proceeds to simulate the premature termination step.
5. $\mathcal{S}_T$ selects $i^\star \in \{1, \ldots, r\}$ with uniform distribution.
6. For each $i \in \{1, \ldots, i^\star - 1\}$ and for each $L \subseteq B \setminus D_0$ s.t. $m - t \leq |L| \leq t$ do
   (a) For each $j \in [m]$, if $j \in L$, then $\mathcal{S}_T$ sets $\widehat{x}_j = x_j$, else, $\mathcal{S}_T$ selects uniformly at random $\widehat{x}_j \in X_n$.
   (b) $\mathcal{S}_T$ sets $\sigma_L^i \leftarrow f_n(\widehat{x}_1, \ldots, \widehat{x}_m)$.
7. The simulator $\mathcal{S}_T$ sends "`proceed`" to all corrupted parties.

Simulating interaction rounds: In each round $1 \leq i \leq r$, the simulator $\mathcal{S}_T$ interacts in three phases with the parties $\{p_j : j \in B \setminus D_0\}$, i.e., the corrupted parties that are active so far:

- **The peeking phase:**
  - If $i = i^\star$, the simulator $\mathcal{S}_T$ sends the set of inputs $\{x_j : j \in B \setminus D_0\}$ to the trusted party computing $\mathcal{F}$ and receives w_S.
  - For each $L \subseteq B \setminus D_0$ s.t. $m - t \leq |L| \leq t$ do
    1. If $i \in \{1, \ldots, i^\star - 1\}$, the simulator $\mathcal{S}_T$ sends the value σ_L^i (prepared in the simulation of the preprocessing phase) to all parties in Q_L (i.e., to the adversary).
    2. Else, if $i \in \{i^\star, \ldots, r\}$, $\mathcal{S}_T$ sends the value w_S to all parties in Q_L (i.e., to the adversary).

- **The abort phase:** Upon receiving an "$\texttt{abort}_j$" message from a party p_j,
  1. $\mathcal{S}_T$ notifies all corrupted parties that party p_j aborted.
  2. $\mathcal{S}_T$ updates $D = D \cup \{j\}$.
  3. If at least $m - t$ parties have aborted so far, that is $|D| \geq m - t$, the simulator $\mathcal{S}_T$ proceeds to simulate the premature termination step.
- **The main phase:** $\mathcal{S}_T$ sends "$\texttt{proceed}$" to all corrupted parties.

Simulating the premature termination step:

- If the premature termination step occurred in round $i = 1$,
  - The simulator $\mathcal{S}_T$ receives a set of inputs $\left\{x'_j : j \in B \setminus D\right\}$ that $\mathcal{A}$ submits to the computation of the dealer.
    If $\mathcal{A}$ does not submit an input on behalf of at least one party p_j, i.e., sends an "$\texttt{abort}_j$" message, then, for each aborted party p_j, the simulator $\mathcal{S}$ notifies all corrupted parties that p_j aborted and updates $D = D \cup \{j\}$.
  - The simulator $\mathcal{S}_T$ sends the set of inputs $\left\{x'_j : j \in B \setminus D\right\}$ to the dealer and receives w_S.
- If the premature termination step occurred in round $1 < i < i^\star$,
  1. Upon receiving an "$\texttt{abort}_j$" message from a party p_j, the simulator $\mathcal{S}_T$ updates $D = D \cup \{j\}$.
  2. The simulator $\mathcal{S}_T$ sends the set of inputs $\{x_j : j \in B\}$ to the trusted party computing $\mathcal{F}$ and receives w_S.
- ($\diamond$ If the premature termination step occurred in round $i^\star \leq i \leq r$, then $\mathcal{S}_T$ already has w_S $\diamond$)
- $\mathcal{S}_T$ sends the value w_S to each party in $\{p_j : j \in B \setminus D_0\}$.

Simulating normal termination: If the last round of the protocol is completed, then $\mathcal{S}_T$ sends w_S to each party in $\{p_j : j \in B \setminus D_0\}$.

At the end of the interaction with $\mathcal{A}$, the simulator outputs the sequence of messages exchanged between the simulator and the corrupted parties.

4.4.2 Proof of the Correctness of the Simulation for $\mathbf{MPCWithDLtd}_r$

In order to prove the correctness of the simulation described in Section 4.4.1, we consider the two random variables from Section 2.2, both of the form (V, C), where V describes a possible view of $\mathcal{A}$, and C describes a possible output of the honest parties (i.e., $C \in Z_n$). The first random variable that describes the real world is $\mathrm{REAL}_{\mathrm{MPCWithDLtd}_r,\mathcal{A}(\mathrm{aux})}(\vec{y}, 1^n)$ – an execution of Protocol $\mathrm{MPCWithDLtd}$, where V describes the view of the adversary $\mathcal{A}$ in this execution, and C is the output of the honest parties in this execution. The second random variable $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S}_T(\mathrm{aux})}(\vec{y}, 1^n)$ describes the ideal world – an execution with the trusted party computing $\mathcal{F}$ (this trusted party is denoted by $T_{\mathcal{F}}$), where V is the view of the simulator $\mathcal{S}_T$ in this execution, and C is the output of the honest parties in this execution. For the rest of this section, we simplify notations and denote the above two random variables by $\mathrm{REAL} = (V_{\mathrm{REAL}}, C_{\mathrm{REAL}})$ and $\mathrm{IDEAL} = (V_{\mathrm{IDEAL}}, C_{\mathrm{IDEAL}})$, respectively.

We consider the probability of a given pair (v, c) according to the two random variables. We compare the two following probabilities: (1) The probability that v is the view of the adversary $\mathcal{A}$ in an execution of

Protocol $\mathrm{MPCWithDLtd}_r$ and c is the output of the honest parties in this execution, where the probability is taken over the random coins of the dealer T. (2) The probability that v is the output of the simulator $\mathcal{S}_T$ in an ideal-world execution with the trusted party $T_{\mathcal{F}}$ and c is the output of the honest parties in this execution, where the probability is taken over the random coins of the simulator $\mathcal{S}_T$ and the random coins of the ideal-world trusted party $T_{\mathcal{F}}$. In Lemma 4.3 we prove the correctness of the simulation by showing that the two random variables are within statistical distance $1/p$.

As the adversary might have some auxiliary information on the inputs of the honest parties and know the value of $f_n(x_1, \ldots, x_m)$, the adversary might be able to deduce that a round is not $i^\star$ if not all the values that it gets are equal to this value (or a possible value for randomized functionalities). Specifically, in the worst case scenario, the adversary knows the inputs of all the honest parties. In the next claim we show a lower bound on the probability that all the values that the adversary obtains in a round $i < i^\star$ of Protocol $\mathrm{MPCWithDLtd}_r$ are all equal to a fixed value.

Claim 4.2. *Let $\mathcal{F}$ be a (possible randomized) functionality computed by Protocol $\mathrm{MPCWithDLtd}_r$ and $d(n)$ and $g(n)$ be the size of the domain and range, respectively. Fix some inputs $x_1, \ldots, x_m$ and w such that $\Pr[f_n(x_1, \ldots, x_m) = w] \geq \epsilon$ for some $\epsilon > 0$. Then, the probability that in a round $i < i^\star$ all the values that the adversary sees are equal to the specific w is at least $(\epsilon/d(n)^m)^{2^t}$.*

Furthermore, if $\mathcal{F}$ is deterministic (thus, $\Pr[f_n(x_1, \ldots, x_m) = w] = 1$), then, this probability is at least $(1/d(n)^m)^{2^t}$.

Proof. We start with the case of a deterministic functionality $\mathcal{F}$. Recall that $x_1, \ldots, x_m$ are the inputs used by the dealer to compute $w = f_n(x_1, \ldots, x_m)$ and $\sigma_L^{i^\star} = w$ for each $L \subseteq [m]$ s.t. $m - t \leq |L| \leq t$. Let L be such that the adversary obtains σ_L^i in round $i < i^\star$. Recall that $\widehat{x}_1, \ldots, \widehat{x}_m$ are the inputs used by the dealer to obtain σ_L^i; that is, $\sigma_L^i = f_n(\widehat{x}_1, \ldots, \widehat{x}_m)$, where $\widehat{x}_j = x_j$ for each $j \in L$ and $\widehat{x}_j$ is selected uniformly at random from X_n for every $j \notin L$. We bound the probability that $\sigma_L^i = w$ by the probability that $\widehat{x}_j = x_j$ for all $j \notin L$. The probability that $\widehat{x}_j = x_j$ is $1/d(n)$. Therefore, the probability that both sets are the same is $(1/d(n))^{m-|L|} > (1/d(n))^m$.

In each round of the protocol, $\mathcal{A}$ obtains the value σ_L^i for each subset Q_L s.t. $L \subseteq [m]$ and $m - t \leq |L| \leq t$; therefore, $\mathcal{A}$ obtains at most 2^t values. For each such two values σ_L^i and $\sigma_{L'}^i$ obtained by $\mathcal{A}$ in round $i < i^\star$, the sets of inputs $\{\widehat{x}_j : j \notin L\}$ and $\{\widehat{x}_j : j \notin L'\}$ are totally independent. Therefore, the probability that all the values that the adversary sees in round $i < i^\star$ are equal to $w = f_n(x_1, \ldots, x_m)$ is at least $(1/d(n)^m)^{2^t}$.

For randomized functionality $\mathcal{F}$, we think of the evaluation of $f_n(\widehat{x}_1, \ldots, \widehat{x}_m)$ as two steps: first $\widehat{x}_j$ is randomly chosen from X_n for every $j \notin L$ and then the randomized functionality is evaluated. Therefore, as $\mathcal{A}$ obtains at most 2^t values in each round $i < i^\star$, the probability that all the values that the adversary sees in each round $i < i^\star$ are equal to the specific w is at least $(1/d(n)^m)^{2^t} \cdot \epsilon^{2^t}$. □

In the next lemma, we prove the correctness of the simulation by using the previous two lemmas.

Lemma 4.3. *Let $\mathcal{F}$ be a (possibly randomized) functionality, $\mathcal{A}$ be a non-uniform polynomial-time adversary corrupting $t < 2m/3$ parties in an execution of Protocol $\mathrm{MPCWithDLtd}$, and $\mathcal{S}_T$ be the simulator described in Section 4.4.1 (where $\mathcal{S}_T$ controls the same parties as $\mathcal{A}$). Then, for every $n \in \mathbb{N}$, for every $\vec{y} \in (X_n)^m$, and for every $\mathrm{aux} \in \{0,1\}^*$*

$$\mathrm{SD}\left(\mathrm{REAL}_{\mathrm{MPCWithDLtd}_r,\mathcal{A}(\mathrm{aux})}(\vec{y}, 1^n), \mathrm{IDEAL}_{\mathcal{F},\mathcal{S}_T(\mathrm{aux})}(\vec{y}, 1^n)\right) \leq 2g(n)d(n)^m / \left(r(n)\right)^{2^t},$$

where $d(n)$ and $g(n)$ are the sizes of the range and the domain of $\mathcal{F}$, respectively, and $r(n)$ is the number of rounds in the protocol.

Furthermore, if $\mathcal{F}$ is deterministic, then, the statistical distance between these two random variables is at most $(d(n)^m)^{2^t}/r(n)$.

Proof. Our goal here is to show that the statistical distance between the above two random variables is at most as described in this lemma. The flow of our proof is as follows.

We first bound the statistical distance between the two random variables by the probability that the adversary $\mathcal{A}$ guesses the special round $i^\star$. We do this by showing that, conditioned on the event that the adversary fails to guess round $i^\star$, the two random variables are identically distributed. Then, we bound the probability of guessing $i^\star$ in time using Lemma 2.6 and Claim 4.2.

Observe that, in the simulation, $\mathcal{S}_T$ follows the same instructions as the trusted party T in Protocol $\mathrm{MPCWithDLtd}_r$, except for two changes. First, $\mathcal{S}_T$ does not compute the output w_S, but rather gets w_S externally from $T_{\mathcal{F}}$. The simulator obtains this value either in the premature termination phase (if $i < i^\star$) or in the peeking stage when $i = i^\star$. The second difference is that in the case of a premature termination, $\mathcal{S}_T$ will always use w_S as its message to the corrupted parties, while T uses the value from round $i^\star - 1$ of the appropriate subset Q_L as its message.

We analyze the probabilities of (v, c) in the two random variables according to weather the premature termination occurred before, during, or after the special round $i^\star$.

Premature termination before round $i^\star$. We argue that in this case, both in the real protocol and in the simulation, the view of $\mathcal{A}$ is identically distributed in the two worlds. $\mathcal{S}_T$ follows the same random process in interacting with $\mathcal{A}$ (before sending the last message in the premature termination) as does T in the real-world execution. The view of the adversary consists of values that are outputs of evaluations of the function f_n on the same input distributions. The adversary does not learn anything about the inputs of the honest parties; hence, its decision to abort does not depend on any new information it obtains during the interaction rounds so far. In addition, in both worlds, the output of the honest parties is the evaluation of the function f_n on the same set of inputs for the active parties and uniformly selected random inputs for the aborted parties.

Premature termination after round $i^\star$ or never occurs. Here v must contain $\sigma_L^{i^\star}$ for some L, which, in the real-world execution, is equal to the output value of all sets for any round $i > i^\star$ (recall that the output value of the honest parties will be determined by one such value), and in the simulation it equals w_S. Thus, in both scenarios, v must be consistent with $i^\star$ and with c; hence, v completely determines C. Again, since $\mathcal{S}_T$ follows the same random process in interacting with $\mathcal{A}$ as does T in the real-world execution the probabilities are the same.

Premature termination in round $i^\star$. This is the interesting case, which causes the statistical distance. In the real world, the output of the honest parties is $\sigma_L^{i^\star - 1}$ for some L, while in the ideal world their output is $w_S \leftarrow f_n(x_1, \ldots, x_m)$. In the first case the output is independent of the adversary's view, while in the second case, the view determines the output. Thus, in this case the probabilities of the views are different. However, we will show that the event of premature termination in round $i^\star$ happens with small probability.

Since the probabilities of (v, c) in the first two cases are equal, the statistical distance between the two random variables is bounded by the probability of the adversary guessing $i^\star$ correctly (before the abort phase of round $i^\star$). That is,

$$\mathrm{SD}(\mathrm{IDEAL}, \mathrm{REAL}) \leq \Pr[\text{Premature termination in round } i^\star]. \tag{2}$$

We next use Lemma 2.6 and Claim 4.2 to bound the probability that the adversary guesses $i^\star$. Let $x_1, \ldots, x_m$ be the inputs obtained by the dealer in the preprocessing phase of Protocol $\mathrm{MPCWithDLtd}_r$. Let p_0 be a parameter specified below. We call an output value w *heavy* if $\Pr[w = f_n(x_1, \ldots, x_m)] > 1/(p_0 \cdot g)$, otherwise, we call w *light* (where the probability is taken over the randomness in computing the functionality). Let S_h and S_ℓ be a partition to heavy and light values, respectively.

Observe that since there are at most g possible values of $f_n(x_1, \ldots, x_m)$, the probability $\Pr[w$ is light$]$, by the union bound, is at most $1/p_0$. Next, by Claim 4.2 where $\epsilon = 1/(p_0 \cdot g)$, the probability that all the values that the adversary sees in round $i < i^\star$ are equal to w is at least $(1/(d^m \cdot p_0 \cdot g))^{2^t}$. By Lemma 2.6 with $\beta = 1/p_0$ and $\alpha = (1/(d^m \cdot p_0 \cdot g))^{2^t}$, the probability that the adversary guesses $i^\star$ is at most $(d^m \cdot p_0 \cdot g)^{2^t}/r + 1/p_0$. We take $p_0 = r^{2^{-t}}/(g \cdot d^m)$ and obtain that the total probability that the adversary guesses $i^\star$ is at most

$$\frac{1}{p_0} + \frac{(d^m \cdot p_0 \cdot g)^{2^t}}{r} \leq 2 \cdot \frac{g \cdot d^m}{r^{2^{-t}}}.$$

Therefore, by (2), the statistical distance between the two random variables in the randomized case is as claimed in the lemma.

The case that $\mathcal{F}$ is deterministic is simpler. By combining Lemma 2.6 where $\beta = 0$ and Claim 4.2 we get that the probability that $\mathcal{A}$ guesses $i^\star$ is at most $(r/d(n)^m)^{2^t}$. By applying Equation (2), we get the bound on statistical distance between the two random variables for the deterministic case as claimed in the lemma.

□

4.4.3 The Simulator for the Protocol with the Dealer for Polynomial Range

Lemma 4.4. *Let $\mathcal{F}$ be a (possibly randomized) functionality. For every non-uniform polynomial-time adversary $\mathcal{A}$ corrupting $t < 2m/3$ parties in an execution of Protocol* $\mathrm{MPCWDPolyRangeLtd}$*, there exists a simulator $\mathcal{S}_T$ in the ideal model that simulates the execution of $\mathcal{A}$ (where $\mathcal{S}_T$ controls the same parties as $\mathcal{A}$). That is, for every $n \in \mathbb{N}$, for every $\vec{y} \in (X_n)^m$, and for every* $\mathrm{aux} \in \{0,1\}^*$

$$\mathrm{SD}\left(\mathrm{REAL}_{\mathrm{MPCWithDLtd}_r, \mathcal{A}(\mathrm{aux})}(\vec{y}, 1^n), \mathrm{IDEAL}_{\mathcal{F}, \mathcal{S}_T(\mathrm{aux})}(\vec{y}, 1^n)\right) < \frac{(2p(n) \cdot g(n))^{2^t}}{r(n)} + \frac{1}{2p(n)},$$

where $g(n)$ is the size of the range of $\mathcal{F}$; with probability $1/(2p(n))$ each value σ_L^i in round $i < i^\star$ is selected uniformly at random from the range, and $r(n)$ be the number of rounds in the protocol.

Proof. The simulators and their proofs for Protocol $\mathrm{MPCWDPolyRangeLtd}$ and Protocol $\mathrm{MPCWithDLtd}$ are similar; we only present (informally) the differences between the two simulators and the two proofs.

The modified simulator. Recall that the protocols $\mathrm{MPCWithDLtd}$ and $\mathrm{MPCWDPolyRangeLtd}$ are different only in Step (3) of the share generation step. In $\mathrm{MPCWDPolyRangeLtd}$, each value σ_L^i prior to round $i^\star$ is chosen with probability $1/(2p)$ as a random value from the range of f_n and with probability $1-1/(2p)$ it is chosen just as in Figure 1. There are two modifications to the simulator. The first modification in the simulator is in Step (6) in the simulation of the preprocessing phase, i.e., in the computation of σ_L^i for $i < i^\star$. The step that replaces Step (6) appears below.

- For each $i \in \{1, \ldots, i^\star - 1\}$ and for each $L \subseteq B \setminus D_0$ s.t. $m - t \leq |L| \leq t$ do
  1. with probability $1/(2p)$, select uniformly at random $z_L^i \in Z_n$ and set $\sigma_L^i = z_L^i$.

2. with the remaining probability $1 - 1/(2p)$,
   (a) For each $j \in [m]$, if $j \in L$, then $\mathcal{S}_T$ sets $\widehat{x}_j = x_j$, else, $\mathcal{S}_T$ selects uniformly at random $\widehat{x}_j \in X_n$.
   (b) $\mathcal{S}_T$ sets $\sigma_L^i \leftarrow f_n(\widehat{x}_1, \ldots, \widehat{x}_m)$.

The second modification is less obvious. Recall that both random variables appearing in the lemma contain the output of the honest parties. In the ideal world, the honest parties always output f_n applied to their inputs. In the real world, in a premature termination in round $i < i^\star$, with probability $1/(2p)$, the honest parties output a random value from the range of f_n. It is hard to simulate the output of the honest parties in first case.[2] We simply modify the simulator such that with probability $1/(2p)$ the simulator returns $\perp$, i.e., it announces that the simulation has failed. The new premature termination step appears below.

Simulating the premature termination step:

- If the premature termination step occurred in round $i < i^\star$,
  - With probability $1/(2p)$, for each $j \in B \setminus D_0$ send "$\texttt{abort}_j$" to the trusted party computing $\mathcal{F}$ and return $\perp$.
  - With the remaining probability $1 - 1/(2p)$, execute the original simulation of the premature termination step (appearing in Section 4.4.1).
- Else ($i \geq i^\star$), execute the original simulation of the premature termination step (appearing in Section 4.4.1).

The modified proof. The proof to the simulator for $\mathrm{MPCWDPolyRangeLtd}$ remains basically the same, except for two changes. We first modify Claim 4.2 below and prove a slightly different claim, which changes the probability of the adversary guessing $i^\star$.

Claim 4.5. *Let $g(n)$ be the size of the range of the (possibly randomized) functionality $\mathcal{F}$ computed by Protocol $\mathrm{MPCWDPolyRangeLtd}_r$ and $w \in Z_n$. Then, the probability that in a round $i < i^\star$ all the values that the adversary sees are equal to w is at least $(1/2p(n) \cdot g(n))^{2^t}$.*

Proof. According to the protocol, there are two different ways to produce each value σ_L^i in round $i < i^\star$: (1) Compute f_n on a set of inputs and a set of uniformly selected values from the domain of the functionality, and (2) Set σ_L^i as a uniformly selected value from the range of the functionality. We ignore the first case. In the second option, with probability $1/2p$, the value σ_L^i is uniformly selected from the range. Hence, the probability that σ_L^i is equal to a specific value is at least $1/(2p \cdot g)$.

It was explained in the proof of Claim 4.2 that in each round of the protocol, $\mathcal{A}$ obtains fewer than 2^t values. Therefore, we conclude that the probability that all the values that $\mathcal{A}$ obtains in round $i < i^\star$ are all equal to w is at least $(1/(2p \cdot g))^{2^t}$. □

By applying the Lemma 2.6 we conclude that the probability of the adversary guessing $i^\star$ correctly in Protocol $\mathrm{MPCWDPolyRangeLtd}_r$ is at most $(2p \cdot g)^{2^t}/r$. In the case of a premature termination in round $i < i^\star$, with probability $1 - 1/(2p)$ in both the ideal world and real world, the value that the honest parties output is the evaluation of f_n on the inputs of the active parties and random inputs for the parties that aborted. However, with probability $1/(2p)$, if premature termination occurs prior to round $i^\star$, the output of the honest

[2] For example, there might not be possible inputs of the corrupted parties causing the honest parties to output such output.

parties in Protocol $\mathrm{MPCWDPolyRangeLtd}_r$ is a random value from the range of f_n; the simulator fails to simulate the execution in this case and outputs $\perp$. Thus,

$$\begin{aligned}
&\mathrm{SD}(\mathrm{IDEAL}, \mathrm{REAL}) \\
&\quad \leq \Pr[\text{Premature termination in round } i^\star] + (1/2p) \cdot \Pr[\text{Premature termination before round } i^\star] \\
&\quad \leq (2p \cdot g)^{2^t}/r + (1/2p).
\end{aligned}$$

Therefore, the statistical distance is as claimed. □

4.5 Proof of Security for the Protocols without the Dealer

4.5.1 The Simulator for Protocol $\mathbf{MPCLtd}_r$

We next prove that Protocol MPCLtd_r is a secure real-world implementation of the (ideal) functionality of Protocol $\mathrm{MPCWithDLtd}_r$. By Lemma 4.3, when $r(n)$ is sufficiently large, Protocol $\mathrm{MPCWithDLtd}_r$ is a $1/p$-secure protocol for $\mathcal{F}$. Thus, together we get that Protocol MPCLtd_r is a $1/p$-secure protocol for $\mathcal{F}$ according to the definition appears in Section 2.3. We analyze Protocol MPCLtd_r in a hybrid model where there are 3 ideal functionalities:

Functionality $\mathbf{ShareGenWithAbortLtd}_r$. This functionality is an (ideal) execution of Functionality $\mathrm{ShareGenLtd}_r$ in the secure-with-abort and cheat-detection model. That is, the functionality gets a set of inputs. If the adversary sends an "$\mathtt{abort}_j$" for at least one corrupted party p_j, then all these messages are sent to the honest parties and the execution terminates. Otherwise, Functionality $\mathrm{ShareGenLtd}_r$ is executed. Then, the adversary gets the outputs of the corrupted parties. Next, the adversary decides whether to halt or to continue: If the adversary decides to continue, it sends a "$\mathtt{proceed}$" message and the honest parties are given their outputs. Otherwise, the adversary sends "$\mathtt{abort}_j$" for at least one corrupted party p_j, and these messages are sent to the honest parties.

Functionality FairMPC. This functionality computes the value $f_n(x_1, \ldots, x_m)$. That is, the functionality gets a set of inputs. If a party p_j sends "$\mathtt{abort}_j$" message then x_j selected from X_n with uniform distribution, computes an output of the randomized functionality f_n for them, and gives it to all parties. When this functionality is executed, an honest majority is guaranteed; hence, the functionality can be implemented with full security (e.g., with fairness).

Functionality Reconstruction. This functionality is described in Figure 4; this functionality is used in the premature termination step in Protocol MPCLtd_r for reconstructing the output value from the shares of the previous round. When this functionality is executed, an honest majority is guaranteed; hence, the functionality can be implemented with full security (e.g., with fairness).

We consider an adversary $\mathcal{A}$ in the hybrid model described above, corrupting $t < 2m/3$ of the parties that engage in Protocol MPCLtd_r. We next describe a simulator $\mathcal{S}$ interacting with the honest parties in the ideal-world via a trusted party $T_{\mathrm{MPCWithDLtd}}$ executing Functionality $\mathrm{MPCWithDLtd}_r$. The simulator $\mathcal{S}$ runs the adversary $\mathcal{A}$ internally with black-box access. Simulating $\mathcal{A}$ in an execution of the protocol, $\mathcal{S}$ corrupts the same subset of parties as does $\mathcal{A}$. Denote by $B = \{i_1, \ldots, i_t\}$ the set of indices of the corrupted parties. At the end of the computation, the simulator outputs a possible view of the real-world adversary $\mathcal{A}$. To start the simulation, $\mathcal{S}$ invokes $\mathcal{A}$ on the set of inputs $\{y_j : j \in B\}$, the security parameter 1^n, and the auxiliary input aux.

Joint input: The security parameter 1^n, the number of rounds in the protocol $r = r(n)$, a bound $t = t(n)$ on the number of corrupted parties, and the set of indices of aborted parties D_0.

Private input: Each party p_j, where $j \notin D_0$, has an input $x_j \in X_n$.

Computing default values

1. For every $j \in D_0$, select x_j with uniform distribution from X_n.
2. Select $i^\star \in [r]$ with uniform distribution and compute $w \leftarrow f_n(x_1, \ldots, x_m)$.
3. For each $1 \leq i < i^\star$, for each $L \subseteq [m] \setminus D_0$ s.t. $m - t \leq |L| \leq t$,
   (a) For each $j \in L$, set $\widehat{x}_j = x_j$.
   (b) For each $j \notin L$, select uniformly at random $\widehat{x}_j \in X_n$.
   (c) Set $\sigma_L^i \leftarrow f_n(\widehat{x}_1, \ldots, \widehat{x}_m)$.
4. For each $i^\star \leq i \leq r$ and for each $L \subseteq [m] \setminus D_0$ s.t. $m - t \leq |L| \leq t$, set $\sigma_L^i = w$.
5. Compute $(K_{\mathrm{sign}}, K_{\mathrm{ver}}) \leftarrow \mathrm{Gen}(1^n)$.

Computing signed shares of the inner secret-sharing scheme

6. For each $i \in \{1, \ldots, r\}$ and for each $L \subseteq [m] \setminus D_0$ s.t. $m - t \leq |L| \leq t$,
   (a) Create shares of σ_L^i in an $|L|$-out-of-$|L|$ secret-sharing scheme for the parties in Q_L. For each party $p_j \in Q_L$, let $S_j^{i,L}$ be its share of σ_L^i.
   (b) Sign each share $S_j^{i,L}$: compute $R_j^{i,L} \leftarrow (S_j^{i,L}, i, L, j, \mathrm{Sign}((S_j^{i,L}, i, L, j), K_{\mathrm{sign}}))$.

Computing shares of the outer secret-sharing scheme

7. For each $i \in [r]$, for each $L \subseteq [m] \setminus D_0$ s.t. $m - t \leq |L| \leq t$, and each $j \in L$, share $R_j^{i,L}$ using a $(t+1)$-out-of-m secret-sharing scheme with respect to p_j as defined in Construction 2.7: compute one masking share $\mathrm{mask}_j(R_j^{i,L})$ and $m-1$ complement shares $(\mathrm{comp}_1(R_j^{i,L}), \ldots, \mathrm{comp}_{j-1}(R_j^{i,L}), \mathrm{comp}_{j+1}(R_j^{i,L}), \ldots, \mathrm{comp}_m(R_j^{i,L}))$.

Signing the messages of all parties

8. For every $1 \leq q \leq m$, compute the message $m_{q,i}$ that $p_q \in P$ broadcasts in round i by concatenating (1) q, (2) i, and (3) the complement shares $\mathrm{comp}_q(R_j^{i,L})$ produced in Step (7) for p_q (for all $L \subseteq [m] \setminus D_0$ s.t. $m - t \leq |L| \leq t$ and all $j \neq q$ s.t. $j \in L$), and compute $M_{q,i} \leftarrow (m_{q,i}, \mathrm{Sign}(m_{q,i}, K_{\mathrm{sign}}))$.

Outputs: Each party p_j such that $j \notin D_0$ receives

- The verification key K_{ver}.
- The messages $M_{j,1}, \ldots, M_{j,r}$ that p_j broadcasts during the protocol.
- p_j's private masks $\mathrm{mask}_j(R_j^{i,L})$ produced in Step (7), for each $1 \leq i \leq r$ and each $L \subseteq [m] \setminus D_0$ s.t. $m - t \leq |L| \leq t$ and $j \in L$.

Figure 2: The initialization functionality $\mathrm{ShareGenLtd}_r$.

Inputs: Each party p_j holds the private input $y_j \in X_n$ and the joint input: the security parameter 1^n, the number of rounds in the protocol $r = r(n)$, and a bound $t = t(n)$ on the number of corrupted parties.

Preliminary phase:

1. $D_0 = \emptyset$
2. If $|D_0| < m - t$,
   (a) The parties in $\{p_j : j \in [m] \setminus D_0\}$ execute a secure-with-abort and cheat-detection protocol computing Functionality $\mathrm{ShareGenLtd}_r$. Each honest party p_j inputs y_j as its input to the functionality.
   (b) If an abort occurred, i.e., at least one party p_j aborts, then for each aborted party p_j, set $D_0 = D_0 \cup \{j\}$. Afterward, goto Step (2).
   (c) Else (no party has aborted), denote $D = D_0$ and proceed to the first round.
3. Otherwise ($|D_0| \geq m - t$), execute premature termination with $i = 1$.

In each round $i = 1, \ldots, r$ **do:**

4. Each party p_j broadcasts $M_{j,i}$ (containing its shares in the outer secret-sharing scheme).
5. For every p_j s.t. $\mathrm{Ver}(M_{j,i}, K_{\mathrm{ver}}) = 0$ or if p_j broadcasts an invalid or no message, then all parties mark p_j as inactive, i.e., set $D = D \cup \{j\}$. If $|D| \geq m - t$, execute premature termination.

Premature termination step

6. If $i = 1$, the active parties use a multiparty secure protocol (with full security) to compute f_n: Each honest party inputs y_j and the input of each inactive party is chosen uniformly at random from X_n. The active parties output the result, and halt.
7. Otherwise,
   (a) Each party p_j reconstructs $R_j^{i-1,L}$, the signed share of the inner secret-sharing scheme produced in Step (6) of Functionality $\mathrm{ShareGenLtd}_r$, for each $L \subseteq [m] \setminus D_0$ s.t. $m - t \leq |L| \leq t$ and $j \in L$.
   (b) The active parties execute a secure multiparty protocol with an honest majority to compute Functionality $\mathrm{Reconstruction}$, where the input of each party p_j is $R_j^{i-1,L}$ for every $L \subseteq [m] \setminus D_0$ s.t. $m - t \leq |L| \leq t$ and $j \in L$.
   (c) The active parties output the output of this protocol, and halt.

At the end of round r**:**

8. Each active party p_j broadcasts the signed shares $R_j^{r,L}$ for each L such that $j \in L$.
9. Let $L \subseteq [m] \setminus D$ be the lexicographical first set such that all the parties in Q_L broadcast properly signed shares $R_j^{r,L}$. Each active party reconstructs the value σ_L^r, outputs σ_L^r, and halts.

Figure 3: The m-party protocol MPCLtd_r for computing $\mathcal{F}$.

Joint Input: The round number i, the indices of inactive parties D, a bound $t = t(n)$ on the number of corrupted parties, and the verification key, K_{ver}.

Private Input of p_j: A set of signed shares $R_j^{i-1,L}$ for each $L \subseteq [m] \setminus D_0$ s.t. $m - t \leq |L| \leq t$ and $j \in L$.

Computation:

1. For each p_j, if p_j's input is not appropriately signed or malformed, then $D = D \cup \{j\}$.
2. Set $L = [m] \setminus D$.
3. Reconstruct σ_L^{i-1} from the shares of all the parties in Q_L.

Outputs: All parties receive the value σ_L^{i-1} (as their output).

Figure 4: Functionality $\mathrm{Reconstruction}$ for reconstructing the output in the premature termination step.

Simulating the preliminary phase:

1. $D_0 = \emptyset$.
2. The simulator $\mathcal{S}$ receives a set of inputs $\{x_j : j \in B \setminus D_0\}$ that $\mathcal{A}$ submits to Functionality $\mathrm{ShareGenWithAbortLtd}_r$.
   If a party p_j for $j \in B \setminus D_0$ does not submit an input, i.e., sends an "$\texttt{abort}_j$" message, then, for each such abort party p_j,
   (a) $\mathcal{S}$ sends "$\texttt{abort}_j$" to the trusted party $T_{\mathrm{MPCWithDLtd}}$.
   (b) $\mathcal{S}$ updates $D_0 = D_0 \cup \{j\}$.
   (c) If $|D_0| < m - t$, then repeat Step (2) of the simulation.
   (d) Otherwise ($|D_0| \geq m - t$), simulate premature termination with $i = 1$.
3. $\mathcal{S}$ prepares outputs for the corrupted parties for Functionality $\mathrm{ShareGenWithAbortLtd}_r$: The simulator $\mathcal{S}$ sets $\sigma_L^i = 0$ for every $L \subseteq [m] \setminus D_0$ s.t. $m - t \leq |L| \leq t$ and for all $i \in \{1, \ldots, r\}$. Then, $\mathcal{S}$ follows Step (1) and Steps (5)–(8) in the computation of Functionality $\mathrm{ShareGenLtd}_r$ (skipping the Steps (2)–(4)) to obtain shares for the parties.[3]
4. For each party p_j s.t. $j \in B \setminus D_0$, the simulator $\mathcal{S}$ sends to $\mathcal{A}$:
   - The verification key K_{ver}.
   - The masking shares $\mathrm{mask}_j(R_j^{i,L})$ for each $i \in \{1, \ldots, r\}$ and for every $L \subseteq [m] \setminus D_0$ s.t. $m - t \leq |L| \leq t$ and $j \in L$.
   - The messages $M_{j,1}, \ldots, M_{j,r}$.
5. If $\mathcal{A}$ sends an "$\texttt{abort}_j$" for at least one party p_j s.t. $j \in B \setminus D_0$ to $\mathcal{S}$, then, for each such aborted party p_j,
   (a) $\mathcal{S}$ sends "$\texttt{abort}_j$" to the trusted party $T_{\mathrm{MPCWithDLtd}}$.
   (b) $\mathcal{S}$ updates $D_0 = D_0 \cup \{j\}$.
   (c) If $|D_0| < m - t$, then repeat Steps (2)–(5) of the simulation.
   (d) Otherwise ($|D_0| \geq m - t$), go to simulating premature termination with $i = 1$.

[3]These shares are temporary and will later be opened for the actual values during the interaction rounds using the properties of Shamir's secret-sharing scheme.

Otherwise ($\mathcal{A}$ sends a "`continue`" message to $\mathcal{S}$),

(a) The simulator $\mathcal{S}$ denotes $D = D_0$.

(b) The simulator sends x_j to $T_{\mathrm{MPCWithDLtd}}$ for every $j \in B \setminus D_0$ (and gets as response a "`proceed`" message).

Simulating interaction rounds:

Let $\mathcal{J}$ be the collection of subsets $L \subseteq B \setminus D_0$ s.t. $m - t \leq |L| \leq t$. I.e., $\mathcal{J}$ is the collection of sets of indices of active corrupted parties after the simulation of the executions of $\mathrm{ShareGenWithAbortLtd}_r$. To simulate round i for $i = 1, \ldots, r$, the simulator $\mathcal{S}$ proceeds as follows:

1. $\mathcal{S}$ gets from the trusted party $T_{\mathrm{MPCWithDLtd}}$ the values that the corrupted parties see. That is, $\mathcal{S}$ gets a bit τ_L^i for each $L \in \mathcal{J}$.[4]
2. The simulator $\mathcal{S}$ selects shares for the inner secret-sharing scheme for corrupted parties: For every $L \in \mathcal{J}$, the simulator $\mathcal{S}$ selects uniformly at random shares of τ_L^i in an $|L|$-out-of-$|L|$ Shamir secret-sharing scheme. Denote these shares by $\left\{X_j^{i,L} : p_j \in Q_L\right\}$.
   For each $p_j \in Q_L$, let $Y_j^{i,L} \leftarrow (X_j^{i,L}, i, L, j, \mathrm{Sign}((X_j^{i,L}, i, L, j), K_{\mathrm{sign}}))$.
3. The simulator $\mathcal{S}$ selects complementary shares for all honest parties: For every $L \in \mathcal{J}$ and for each $j \in B \setminus D_0$,
   (a) S calculates $\alpha_j = \mathrm{mask}_j(R_j^{i,L}) \oplus Y_j^{i,L}$.
   (b) S selects uniformly at random $m - t$ shares of α_j uniformly at random over all possible selections of $m - t$ shares that are shares of α_j together with the $|B \setminus D_0| - 1$ shares
   $$\left\{\mathrm{comp}_q(R_j^{i,L}) : q \in B \setminus (D_0 \cup \{j\})\right\}$$
   produced in Step (3) in the simulation of the preliminary phase.
   (This is possible according to the property of Shamir's scheme.)
   Denote by $\mathrm{comp}_q(Y_j^{i,L})$ the complementary share that $\mathcal{S}$ selects for the honest party p_q for a party p_j s.t. $j \in (B \setminus D_0) \cap L$, where $L \in \mathcal{J}$.
4. For party p_j and a subset $L \notin \mathcal{J}$, let $\mathrm{comp}_q(R_j^{i,L})$ be the complementary share that was produced in Step (3) in the simulation of the preliminary phase, i.e., $\mathrm{comp}_q(R_j^{i,L})$.
5. Construct signed messages $m'_{q,i}$ for each honest party p_q in round i by concatenating:
   (a) q.
   (b) The round number i.
   (c) The complement shares that were described in Step (4) above.
   (d) The complement shares $\mathrm{comp}_q(Y_j^{i,L})$ for all $L \in \mathcal{J}$ and for all $j \in L$ produced in Step (3) for p_q.
   Then, $\mathcal{S}$ signs $m'_{q,i}$, i.e., $\mathcal{S}$ computes $M'_{q,i} \leftarrow (m'_{q,i}, \mathrm{Sign}(m'_{q,i}, K_{\mathrm{sign}}))$.
6. The simulator $\mathcal{S}$ sends all the messages $M'_{q,i}$ on behalf of each honest party p_q to $\mathcal{A}$.

[4] In Steps (2)–(5), the simulator $\mathcal{S}$ constructs the messages of the honest parties in order to allow the corrupted parties in each $L \in \mathcal{J}$ to reconstruct τ_L^i.

7. For every $j \in B \setminus D_0$ s.t. $\mathcal{A}$ sends an invalid or no message on behalf of p_j, the simulator $\mathcal{S}$ sends "$\texttt{abort}_j$" to $T_{\text{MPCWithDLtd}}$:
   (a) $D = D \cup \{j\}$.
   (b) If $|D| \geq m - t$ go to premature termination step.
   (c) Otherwise, the simulator $\mathcal{S}$ proceeds to the next round.

Simulating the premature termination step:

- If $i = 1$, then S simulates $\mathcal{A}$'s interaction with Functionality FairMPC as follows:
  1. S receives from $\mathcal{A}$ the inputs of the active corrupted parties.
  2. For every $j \in B \setminus D$: If p_j does not send an input, then $\mathcal{S}$ sends "$\texttt{abort}_j$" to $T_{\text{MPCWithDLtd}}$ else, $\mathcal{S}$ sends p_j's input to $T_{\text{MPCWithDLtd}}$.
- If $i > 1$, then S simulates $\mathcal{A}$'s interaction with Functionality Reconstruction as follows:
  1. S receives from $\mathcal{A}$ the inputs of the active corrupted parties, i.e., p_j s.t. $j \in B \setminus D$.
  2. If an active corrupted party p_j does not send an input, or its input is not appropriately signed or malformed, then $\mathcal{S}$ sends "$\texttt{abort}_j$" to $T_{\text{MPCWithDLtd}}$.
- $\mathcal{S}$ gets from $T_{\text{MPCWithDLtd}}$ a value σ and sends it to $\mathcal{A}$.
- The simulator $\mathcal{S}$ outputs the sequence of messages exchanged between $\mathcal{S}$ and the adversary $\mathcal{A}$ and halts.

Simulating normal termination at the end of round r:

1. The simulator gets w from the trusted party $T_{\text{MPCWithDLtd}}$.
2. $\mathcal{S}$ constructs all the signed shares of the inner secret-sharing scheme for each $L \subseteq [m] \setminus D_0$ s.t. $m - t \leq |L| \leq t$ and for each honest party $p_j \in Q_L$ as follows.
   For each $L \notin \mathcal{J}$, the simulator S selects uniformly at random $|L \setminus B|$ shares of w uniformly at random over all possible selections of $|L \setminus B|$ shares that together with the $|L \cap B|$ given shares $\left\{R_j^{i,L} : j \in B\right\}$ (produced in Step (2) in the simulation of the preliminary phase) are a sharing of w in an $|L|$-out-of-$|L|$ secret-sharing scheme.
   (This is possible according to the property of Shamir's scheme.)
   Denote these shares by $\left\{X_j^{r,L}\right\}$.

   For each share $X_j^{r,L}$, the simulator concatenates the corresponding identifying details, and signs them to obtain: $Y_j^{r,L} \leftarrow (X_j^{r,L}, r, L, j, \text{Sign}((X_j^{r,L}, r, L, j), K_{\text{sign}}))$.
3. For each honest party p_j, the simulator $\mathcal{S}$ sends to $\mathcal{A}$ the shares $Y_j^{r,L}$ for all subsets L, such that $p_j \in Q_L$.
4. The simulator $\mathcal{S}$ outputs the sequence of messages exchanged between $\mathcal{S}$ and the adversary $\mathcal{A}$ and halts.

4.6 Proving the Correctness of Protocol MPCLtd_r and Protocol MPCPolyRangeLtd_r

We claim that Protocol MPCLtd_r is a secure implementation of the (ideal) functionality of the dealer in Protocol MPCWithDLtd_r. That is,

Lemma 4.6. *Let $t < 2m/3$. If enhanced trap-door permutations exist, then Protocol MPCLtd_r, presented in Section 4.2, is a computationally secure implementation (with full security) of the dealer functionality in Protocol $\mathrm{MPCWithDLtd}_r$.*

In [5], a similar framework to the one used in this paper is used: first a protocol with a dealer for the coin-tossing problem is presented, and then, a real-world protocol that is a computationally secure implementation (with full security) of the dealer functionality is described. In [5], a simulator for this protocol is given; this simulator is similar to the simulator described in Section 4.5.1. Then a full proof for the simulator is provided. As the proof is very similar to the proof of our simulator, it is omitted.

To conclude the proof, as $\mathrm{MPCWithDLtd}_r$ is a $1/p$-secure implementation of $\mathcal{F}$ and MPCLtd_r is a secure implementation of the (ideal) functionality of the dealer in Protocol $\mathrm{MPCWithDLtd}_r$, by the composition theorem of Canetti [9] we conclude that MPCLtd_r is a $1/p$-secure implementation of $\mathcal{F}$. That is, Theorem 3 is proved.

Next, we claim that $\mathrm{MPCPolyRangeLtd}_r$ is a secure implementation of the (ideal) functionality of the dealer in Protocol $\mathrm{MPCWDPolyRangeLtd}_r$. That is,

Lemma 4.7. *Let $t < 2m/3$. If enhanced trap-door permutations exist, then Protocol $\mathrm{MPCPolyRangeLtd}_r$, described in Section 4.3, is a computationally-secure implementation (with full security) of the dealer functionality in Protocol $\mathrm{MPCWDPolyRangeLtd}_r$.*

Proof. Recall that the only difference between Protocol MPCLtd_r and Protocol $\mathrm{MPCPolyRangeLtd}_r$ is in the way that the values that the parties see prior to round $i^\star$ are produced, i.e., the difference is in Functionality $\mathrm{ShareGenLtd}_r$. Specifically, in Section 4.3 we presented a modification in Step (3) in Functionality $\mathrm{ShareGenLtd}_r$ in order to get Protocol MPCLtd_r from Protocol $\mathrm{MPCPolyRangeLtd}$. Now, observe that the simulator presented above does not refer to Step (3) of Functionality $\mathrm{ShareGenLtd}_r$ in any step. Therefore, the simulator presented in Section 4.5.1 for Protocol MPCLtd_r is also a simulator for Protocol $\mathrm{MPCPolyRangeLtd}_r$. □

Claim 4.5 and Lemma 4.7 imply Theorem 4.

5 Protocols for any Number of Corrupted Parties

In this section we describe our protocols that are secure when the adversary can corrupt any number of parties. We start with a protocol that assumes that either the functionality is deterministic and the size of the domain is polynomial, or that the functionality is randomized and the size of both the domain and range of the functionality are polynomial. We then present a modification of the protocol that is $1/p$-secure for (possibly randomized) functionalities if the size of the range is polynomial (even if the size of the domain of $\mathcal{F}$ is not polynomial). The first protocol is more efficient for deterministic functionalities with polynomial-size domain. Furthermore, the first protocol has full correctness, while in the modified protocol, correctness is only guaranteed with probability $1 - 1/p$.

Unlike Section 4, we directly describe our protocol without any trusted dealer. In Section 5.1 we present the protocol and in Sections 5.2–5.3 we prove its security. The formal description of the protocol appears in Figures 6–11. In Section 5.4, we present a modification of the protocol that is $1/p$-secure if the size of the range is polynomial (even if the size of the domain of f is not polynomial). The basic structure of both protocols appears in Figure 5.

For simplicity of the presentation, in the rest of this section, we assume that in the interaction phase of the protocol the adversary is a *fail-stop* adversary. That is, all parties follow the protocol with one exception:

the corrupted parties may abort the computation at any time. For completeness, in Section 5.1.1 we describe how to get rid of this assumption using signatures.

5.1 The m-Party Protocol for Polynomial-Size Domain

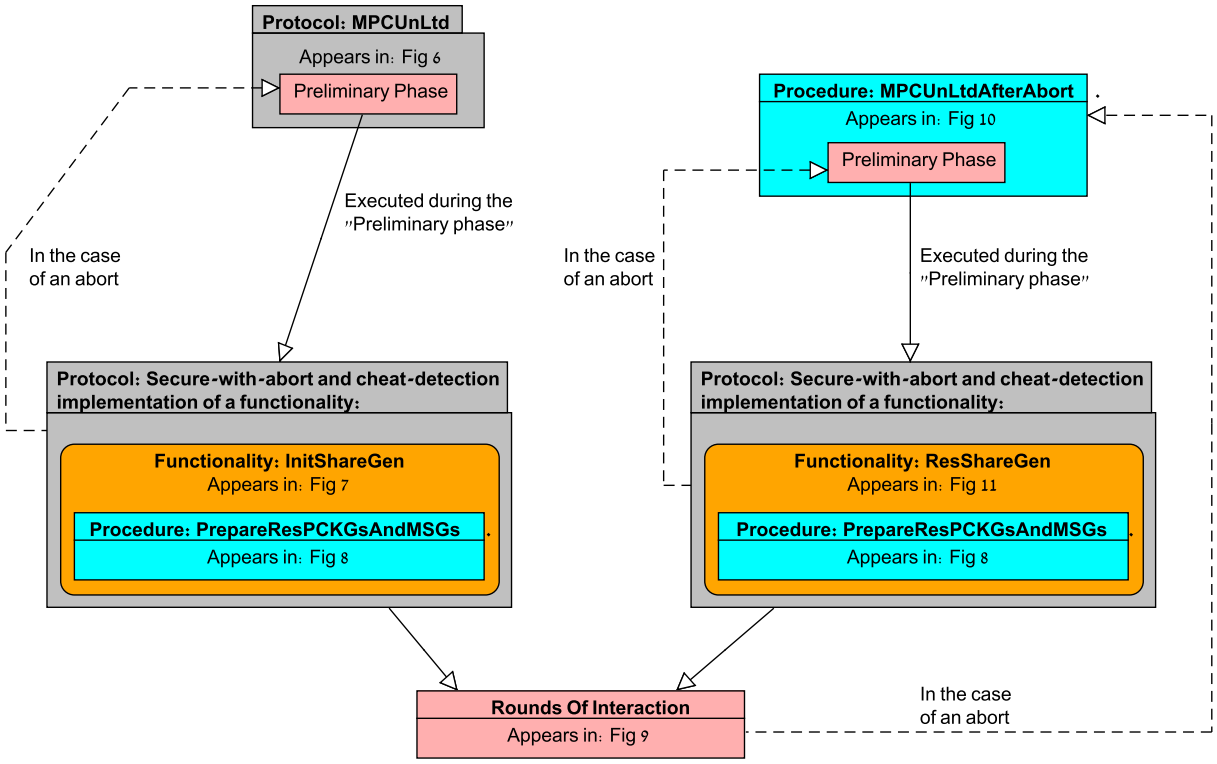

Figure 5: The basic flow of the m-party protocol.

Recall that the protocol that appears in Section 4 has two phases: The first phase is a preliminary phase in which the parties compute a given functionality in a secure-with-abort with cheat-detection manner. The output of this functionality for each party p_j includes the messages that p_j has to broadcast throughout the second phase – called the interaction phase. Thus, by the end of the first phase, the parties are prepared for any possible execution of the interaction phase, i.e., the honest parties can complete r rounds of interactions regardless of any possible abort of any subset in any round during the protocol. However, the protocol in this section has two basic phases that have slightly different roles.

Informally, the preliminary phase gives the parties less "direct information" but yet enough information for a single execution of the interaction phase. To overcome the gap, the combination of a single preliminary phase followed by a single interaction phase might be executed several times, according to possible aborts of the adversary. That is, in the first preliminary phase the parties get messages to be sent until the first abort. If a party aborts, the parties execute another preliminary phase (using information from the previous preliminary phase) to prepare messages until the next abort, and this process continues until all interaction rounds are completed.

We next sketch the basic flow of the resulting protocol. At first, the parties compute Functionality $\mathrm{InitShareGen}_r$ in a secure-with-abort with cheat-detection model. This functionality gets the parties inputs

and gives in return a set of messages for executing the interaction phase. These messages include additional information which allows the parties to recover from any potential abort. We call this additional information – "resumption packages". In the case of an abort in the interaction phase, the parties compute Functionality $\mathrm{ResShareGen}_r$ in a secure-with-abort with cheat-detection model; this functionality takes as an input the corresponding shares of the "resumption packages" that the parties hold and returns to each active party as an output a set of messages, which are of the same type as described before. These messages are aimed at executing an additional r-rounds of interactions phase. They are also include, in some sense recursively, a set of "resumption packages", which allow the parties to recover from any potential abort in a later stage of the protocol. It is important to emphasize that both functionalities $\mathrm{ResShareGen}_r$ and $\mathrm{InitShareGen}_r$ return the same output – a set of messages that include the "resumption packages"; however, the latter functionality takes as an input the original input of the parties, while the former functionality takes as an input the "resumption packages" that the parties keep learning in each (successful) round of the interaction phase.

We next describe the protocol in more detail. For simplifying the explanation we assume that $m = 3$, i.e., there are 3 parties, any 2 of them can be corrupted. In the first step, the parties execute Functionality $\mathrm{InitShareGen}_r$ in a secure-with-abort with cheat-detection model. As a result, for each round i and for each subset $L \subset \{1, \ldots, 3\}$, a value w^i_L is chosen similarly to the way the values in the protocols in Section 4 are chosen. Jumping ahead, this value is related to the parties $\{p_j : j \in L\}$ and are used if all other parties abort in round $i + 1$. Let us explain how these values are selected. As in our previous protocol, there is a special round, called $i^\star$, which is chosen with a uniform distribution from $\{1, \ldots, r\}$. Prior to round $i^\star$, the values that are chosen for each subset depend only on the inputs of the subset: random inputs are chosen for the parties not in the subset and the function f_n is computed with the inputs of the subset and the random inputs for other parties. Starting from round $i^\star$, the value of each subset is the output w of f_n on the inputs of all parties.

If no party aborts during the protocol, then each party p_j outputs the value $w^r_{\{j\}}$. If 2 corrupted parties abort in some round i, then the third party p_j outputs the value $w^{i-1}_{\{j\}}$. The difficult case is when one party, say p_3, aborts in some round i. In this case one of the active parties p_1 or p_2 might be corrupted. Thus, p_1 and p_2 execute a 2-party r-round $O(1/\sqrt{r})$-secure version of the above mentioned 3-party protocol (this is the 2-party protocol of [24]). As we mentioned before, the protocol is "not given" to the parties in a direct way, i.e., the parties do not hold a set of messages for all the upcoming rounds; however, p_1 and p_2 hold the following information (in a secret shared way), which can turn into an r-round protocol between p_1 and p_2: (1) $\{x_1, x_2\}$ – the inputs of the remaining parties. (2) $i^\star_{\{1,2\},i}$ – the special $i^\star$ value of this (future) protocol. (3) $w^i_{\{1,2\}}$ – the output value of this (future) protocol. We call this information "resumption packages" and we denote it by $\mathrm{S}_{\{1,2\},i}$. The string $\mathrm{S}_{\{1,2\},i}$ cannot be given directly to p_1 and p_2; thus, it is being kept in a secret shared way between the two parties.

We next explain how exactly the "resumption packages" are used to produce messages for a two-party r-round protocol. The "resumption package" for p_1, p_2 was generated for the scenario that p_3 is corrupted and at least one of p_1, p_2 is honest. We next consider the scenario that they are both corrupted (and p_3 is honest). In each round i they can "open" the "resumption packages" as if p_3 has aborted. If $i^\star_{\{1,2\},i} > 1$, they conclude that $i < i^\star$. Thus, they can determine $i^\star$ and bias the output of the protocol with a high probability. To overcome this problem, we modify the way that $i^\star_{L,i}$ is chosen prior to round $i^\star$: with probability p, to be determined later, set $i^\star_{L,i} = 1$, and with the remaining probability choose it at random from $\{1, \ldots, r\}$. Notice that the "resumption package" that can be opened by p_1, p_2 in the case $i^\star_{L,i} = 1$ looks like the "resumption package" in rounds starting from $i^\star$; thus, by Lemma 2.6, the probability that the corrupted parties guess $i^\star$ is $O(1/pr)$. However, a corrupted p_2 can bias the protocol by guessing $i^\star_{L,i} = 1$

and aborting in round 1 of the two-party protocol. This can cause an additional bias of at most p. Choosing $p = O(1/\sqrt{r})$, the total bias of the protocol is $O(1/pr) + p = O(1/\sqrt{r})$; that is, the resulting protocol is $O(1/\sqrt{r})$-secure.

The m-party protocols tolerating up to $m-1$ corrupted parties use the same structure as our 3-party protocols. In a preliminary phase, $i^\star$ and values w^i_L for every $L \subset \{1,\ldots,m\}$ are chosen as above. If some set of parties aborts in some round i, then the remaining parties indexed by $|L|$ execute the $|L|$-party protocol that they produce from the "resumption packages", where if $i \geq i^\star$ then it uses $i^\star_{L,i} = 1$, and if $i < i^\star$ then $i^\star_{L,i} = 1$ with probability $r^{-1/\left(2^{|L|}-|L|\right)}$ and $i^\star_{L,i}$ is random otherwise. In this $|L|$-party protocol, if a party aborts, the remaining L' parties execute an $|L'|$-party protocol that they produce from the "resumption packages" (again with its special round being set to 1 with some probability), and so on.

In the above, we only sketched the protocol for 3-parties. The formal description for m-parties appears in Figures 6–11. The basic structure of both protocols appears in Figure 5. A small modification of this protocol, yielding a $\mathbf{1/p}$-secure protocol for polynomial range, appears in Section 5.4. Basically, we use the same trick used in Section 4.3: With some small probability a value given to the set is chosen from the range prior to $\boldsymbol{i^\star}$ in the 3-party interaction and prior to $\boldsymbol{i^\star_{L,i}}$ in the two parties' protocols. The proof that both protocols are $\mathbf{1/p}$-secure appears in Sections 5.2–5.3.

Inputs: Private inputs: Each party $\boldsymbol{p_\ell}$ holds the private input $\boldsymbol{y_\ell} \in \boldsymbol{X_n}$. Joint input: The security parameter $\mathbf{1^n}$ and the number of rounds in the protocol $\boldsymbol{r = r(n)}$.

Preliminary phase:

1. $\boldsymbol{L = \{1,\ldots,m\}}$.
2. Each party $\boldsymbol{p_\ell}$ computes its default value $\boldsymbol{w^0_{\{\ell\}}}$:
   (a) $\boldsymbol{p_\ell}$ sets $\boldsymbol{\widehat{x}_\ell = y_\ell}$ and for every $\boldsymbol{j \in \{1,\ldots,m\} \setminus \{\ell\}}$, selects $\boldsymbol{\widehat{x}_j}$ with uniform distribution from $\boldsymbol{X_n}$.
   (b) $\boldsymbol{p_\ell}$ sets $\boldsymbol{w^0_{\{\ell\}} \leftarrow f_n(\widehat{x}_1,\ldots,\widehat{x}_m)}$.
3. The parties execute a secure-with-abort and cheat-detection protocol computing Functionality $\mathbf{InitShareGen}$ (see Figure 7). Each honest party $\boldsymbol{p_\ell}$ inputs $\boldsymbol{y_\ell}$ as its input to the functionality. Joint input for the protocol is: $\boldsymbol{L}$, $\mathbf{1^n}$, and $\boldsymbol{r}$.
4. If an abort has occurred during the execution of the protocol executed in the previous step, that is, the output of the honest parties is "$\texttt{abort}_j$" for at least one index $\boldsymbol{j}$, then,
   (a) For every $\boldsymbol{p_j}$ that aborted, all parties mark $\boldsymbol{p_j}$ as inactive, i.e., remove $\boldsymbol{j}$ from $\boldsymbol{L}$.
   (b) If only $\boldsymbol{p_\ell}$ is active, then $\boldsymbol{p_\ell}$ outputs $\boldsymbol{w^0_{\{\ell\}}}$ and halts.
   (c) Else, go to Step (3).
5. Else, if no abort has occurred, execute the rounds of interaction as described in Figure 9 where each party holds its private default value, a set of messages, and private shares for $\boldsymbol{r}$ rounds of interaction.

Figure 6: The preliminary phase of the $\boldsymbol{m}$-party protocol $\mathbf{MPCUnLtd_r}$ for computing $\boldsymbol{\mathcal{F}}$.

Inputs: Private inputs: Each party p_ℓ holds $x_\ell \in X_n$. The joint input: The set of indices of the active parties L, the security parameter 1^n, and the number of rounds $r = r(n)$.

Computing default values and construct the protocol

1. Select $i^\star \in \{1, \ldots, r\}$ with uniform distribution.
2. For each $\ell \in L$, set $\widehat{z}_\ell = x_\ell$, for each $j \in \{1, \ldots, m\} \setminus L$ select uniformly at random $\widehat{z}_j \in X_n$, and set $\sigma \leftarrow f_n(\widehat{z}_1, \ldots, \widehat{z}_m)$.
3. Compute Procedure $\mathbf{PrepareResPCKGsAndMSGs}$ (see Figure 8) with σ,$i^\star$,L,$\{x_\ell\}_{\ell \in L}$, and r and return its outputs.

Output of party p_ℓ: The output from procedure $\mathbf{PrepareResPCKGsAndMSGs}$, i.e., p_ℓ gets a set of messages and private shares for the execution of the r-rounds of interaction.

Figure 7: Functionality $\mathbf{InitShareGen}_r$.

5.1.1 Signatures and Verification against any Computational Bounded Adversary

Protocol $\mathbf{MPCUnLtd}_r$ assumes a fail-stop adversary, i.e., the adversary is only allowed to abort the computation by not sending any message in a certain round. Such an adversary simplifies the presentation of the protocol. By several changes to the protocol, the protocol will we be $1/p$-secure even in the presence of any computational bounded adversary, while allowed to attack the protocol in any arbitrary manner. Two major changes should be made:

Adding signatures In the procedure $\mathbf{PrepareResPCKGsAndMSGs}$, a signing key is produced. Using this key, all the messages $M_{i,j}$ and all resumption packages $\mathbf{S}_{J,i}$ are signed by the signing key. These signatures prevent the adversary from forging messages.

A distributed mechanism for achieving an agreement. Protocol $\mathbf{MPCUnLtd}_r$ uses two functionalities: $\mathbf{InitShareGen}_r$ and $\mathbf{ResShareGen}_r$. Recall that in both of them, one of the joint inputs is the set of active parties indexed by L. This set is updated during an execution of the protocol according to possible aborts of parties. As we use cheat detection, the honest parties update the set of L in each step of the protocol. Each party uses its private set of indices L as an input to the relevant protocols. However, as corrupted parties can input an incorrect set L, we need to modify the definition of the functionality. As there is no honest majority, the functionality cannot use the most frequent set L. The key-observation to overcome this problem is that although the corrupted parties might input a partial or incorrect set of indices, the honest parties agree on the same set. This observation yields the following modification to the protocols: A verification step should be added to the functionalities $\mathbf{InitShareGen}_r$ and $\mathbf{ResShareGen}_r$ that consists of a check that all the sets inputed by the parties are equal. If not, the functionality computes for each party p_j the minimal index of party p_ℓ whose input L does not agree with the input L of p_j and returns the index ℓ to p_j. Party p_j removed ℓ from the list of active parties. As a result, the honest parties always keep track of the set of active parties and execute the protocol that implements the relevant functionality with the updated set.

We next prove the security of Protocol $\mathbf{MPCUnLtd}_r$. In Section 5.2 we present the simulator for Protocol $\mathbf{MPCUnLtd}_r$ and in Section 5.3 we prove its correctness.

Inputs: The target value σ, the value $i^\star$, the set of active parties indexed by L, the inputs of the active parties $\{x_\ell\}_{\ell \in L}$, and the number of rounds r.

Computing default values for "resumption packages":

1. For each $1 \leq i < i^\star$ and for each $J \subset L$,
   (a) With probability $r^{-1/\left(2^{|L|}-|L|\right)}$, set $i^\star_{J,i} = 1$.
   With the remaining probability, select $i^\star_{J,i} \in \{1, \ldots, r\}$ with uniform distribution.
   (b) For each $\ell \in J$ set $\widehat{z}_\ell = x_\ell$, for each $j \in \{1, \ldots, m\} \setminus J$ select $\widehat{z}_j \in X_n$ uniformly at random, and set $w^i_J \leftarrow f_n(\widehat{z}_1, \ldots, \widehat{z}_m)$.
2. For each $i^\star \leq i \leq r$ and for each $J \subset L$ set $i^\star_{J,i} = 1$ and $w^i_J = \sigma$.

Computing the shares of the "resumption packages":
For each $1 \leq i \leq r$ and for each $J \subset L$ s.t. $|J| \geq 2$,

1. Compute the "resumption package" $\mathbf{S}_{J,i}$ of the set J for the case that parties in $L \setminus J$ abort in round $i+1$ consisting of:
   - $\{x_\ell\}_{\ell \in J}$ – The inputs of the (future) active parties.
   - $i^\star_{J,i}$ – the special $i^\star$ value for the (future) protocol.
   - w^i_J – the output value for the (future) protocol.
2. Share $\mathbf{S}_{J,i}$ in an $|J|$-out-of-$|J|$ secret-sharing scheme for the parties $\{p_\ell : \ell \in J\}$. For each $\ell \in J$, let $[\mathbf{S}_{J,i}]_\ell$ be the share of p_ℓ of the string $\mathbf{S}_{J,i}$.

Computing messages for uninterrupted interaction:

Computing the secrets that the party learn in a round:
For each $1 \leq i \leq r$ and for each $\ell \in L$, construct the string $s_{i,\ell}$ (p_ℓ learns $s_{i,\ell}$ in round i) consisting of:

- The default value $w^i_{\{\ell\}}$.
- The shares $[\mathbf{S}_{J,i}]_\ell$ of party p_ℓ for each $J \subset L$ s.t. $|J| \geq 2$ and $\ell \in J$.

Computing the messages that the parties broadcasts in a round
For each $1 \leq i \leq r$ and for each $\ell \in L$,

1. Share the string $s_{i,\ell}$ in an $|L|$-out-of-$|L|$ secret-sharing scheme. For each $\ell' \in L$, let $[s_{i,\ell}]_{\ell'}$ be the share of $p_{\ell'}$ of the secret $s_{i,\ell}$.
2. Set $M_{i,\ell} \leftarrow \left([s_{i,\ell'}]_\ell\right)_{\ell \neq \ell'}$ (the message that p_ℓ has to broadcast in round i).

Outputs: Each party p_ℓ receives

- The messages $M_{1,\ell}, \ldots, M_{r,\ell}$ that p_ℓ broadcasts during the protocol.
- The private shares $[s_{1,\ell}]_\ell, \ldots, [s_{r,\ell}]_\ell$ of p_ℓ.

Figure 8: Procedure $\mathbf{PrepareResPCKGsAndMSGs}$.

Inputs: Private inputs: Each party p_ℓ holds a private default value $w^0_{\{\ell\}}$, a set of messages $M_{1,\ell}, \ldots, M_{r,\ell}$, and a set of private shares $[s_{1,\ell}]_\ell, \ldots, [s_{r,\ell}]_\ell$.
The joint input: The set of active parties indexed by L, the security parameter 1^n, and the number of rounds in the protocol r.

Rounds of interaction: `In each round` $i = 1, \ldots, r$ `do:`

1. Each party p_ℓ broadcasts $M_{i,\ell}$.
2. If an abort occurred, i.e., at least one active party did not send a message, then,
   (a) For every p_j that aborts, all parties mark p_j as inactive, i.e., remove j from L.
   (b) If only p_ℓ is active, then p_ℓ reconstructs the value $w^{i-1}_{\{\ell\}}$, outputs it, and halts.
   (c) Else (at least 2 parties are active),
      i. Each party p_ℓ extracts $w^{i-1}_{\{\ell\}}$ from $s_{i,\ell}$ and updates its default value: $w^0_{\{\ell\}} = w^{i-1}_{\{\ell\}}$.
      ii. The (active) parties execute the procedure $\mathbf{MPCUnLtdAfterAbort}$ (see Figure 10), where each party p_ℓ holds as its private input the set of its shares of the "resumption packages" – $[\mathbf{S}_{J,i-1}]_\ell$ for each $J \subseteq L$ s.t. $\ell \in L'$ and its default value $w^0_{\{\ell\}}$. Joint input: L, 1^n, and r.

`At the end of round` r**:** Each active party p_ℓ reconstructs the value $w^r_{\{\ell\}}$, outputs it, and halts.

Figure 9: The interaction rounds of Protocol $\mathbf{MPCUnLtd}_r$.

Inputs: Private inputs: Each party p_ℓ holds a set of its shares of the "resumption packages" – $\mathbf{S}_{J,\ell}$ for each $J \subseteq L$ s.t. $\ell \in J$ and the default value $w^0_{\{\ell\}}$. Joint input: The set of active parties indexed by L, the security parameter 1^n, and the number of rounds in the protocol r.

Preliminary phase:

1. The parties execute a secure-with-abort and cheat-detection protocol for computing Functionality $\mathbf{ResShareGen}$ (see Figure 11). Each party p_ℓ inputs its share of $\mathbf{S}_{L,\ell}$ – the "resumption package" for L. In addition, the protocol gets the joint inputs: L, 1^n, and r (see Section 5.1.1 for more on this).
2. If an abort has occurred during the execution of the protocol executed in Step (1), that is, the output of the honest parties is "`abort`$_j$" for at least one index j, then,
   (a) For every p_j that aborts, all parties mark p_j as inactive, i.e., remove j from L.
   (b) If only p_ℓ is active, then the party p_ℓ outputs the value $w^0_{\{\ell\}}$ and halts.
   (c) Else, go to Step (1).
3. Else, if no abort has occurred, execute the rounds of interaction as described in Figure 9 where each party holds its private default value and a set of messages and private shares for the r-rounds of interaction.

Figure 10: Procedure $\mathbf{MPCUnLtdAfterAbort}$.

Inputs: Private input: Each party p_ℓ holds its share in the "resumption package" – $\mathbf{S}_\ell$.
Joint input: The set of active parties indexed by L, the security parameter 1^n, and the number of rounds r.

Computing default values and construct the protocol:

1. Reconstruct the following values from the shares in $\{\mathbf{S}_\ell\}_{\ell \in L}$:
   - x_ℓ for each $\ell \in L$ – The input of each active party p_ℓ
   - $i^\star$ – The special round $i^\star$ for the set L
   - σ – the prescribed output of L
2. Execute the procedure $\mathbf{PrepareResPCKGsAndMSGs}$ (see Figure 8) with σ,$i^\star$,L,$\{x_\ell\}_{\ell \in L}$ and r and return its outputs.

Output party p_ℓ: The output from procedure $\mathbf{PrepareResPCKGsAndMSGs}$, i.e., p_ℓ gets a set of messages and private shares for the execution of the r-rounds of interaction.

Figure 11: Functionality $\mathbf{ResShareGen}_r$.

5.2 The Simulator for Protocol $\mathrm{MPCUnLtd}_r$

We analyze Protocol $\mathbf{MPCUnLtd}_r$ in a hybrid model with the following blackbox implementations of functionalities in the secure-with-abort and cheat-detection model.

- $\mathbf{InitShareGenWithAbort}_r$ – an implementation of Functionality $\mathbf{InitShareGen}_r$.
- $\mathbf{ResShareGenWithAbort}_r$ – an implementation of Functionality $\mathbf{ResShareGen}_r$.

That is, both implementations get a set of inputs according to the descriptions appears in Figure 7 and Figure 11, respectively. If the adversary sends "`abort`$_j$" for at least one corrupted party p_j, then these messages are sent to the honest parties and the execution of the corresponding functionality terminates. Otherwise, Functionality $\mathbf{InitShareGen}_r$ or Functionality $\mathbf{ResShareGen}_r$ is executed, respectively. In the next step, the adversary gets the outputs of the corrupted parties. Next, the adversary decides whether to halt or to continue: If the adversary decides to continue, it sends a "`proceed`" message and the honest parties are given their outputs. Otherwise, the adversary sends "`abort`$_j$" for at least one corrupted party p_j, and these messages are sent to the honest parties.

We consider an adversary $\mathcal{A}$ in the hybrid model described above. We denote by B the set of indices of corrupted parties. To start the simulation, $\mathcal{S}$ invokes $\mathcal{A}$ on the set of inputs $\{y_\ell : \ell \in B\}$, the security parameter 1^n, and the auxiliary input $\mathbf{aux}$. A brief description of the simulator appears in Figure 12. A formal description of the simulation follows:

Simulating the preliminary phase of Protocol $\mathrm{MPCUnLtd}_r$:

`/* Protocol` $\mathbf{MPCUnLtd}_r$ `appears in Figure 6. */`

1. $L = \{1, \ldots, m\}$.
2. The simulator $\mathcal{S}$ simulates the interaction of $\mathcal{A}$ with $\mathbf{InitShareGenWithAbort}_r$ as follows:
   `/* This part consists of two main steps: (1)` $\mathcal{S}$ `sees the inputs of the corrupted parties to` $\mathbf{InitShareGenWithAbort}_r$`, and, (2)` $\mathcal{S}$ `produces the outputs for the active corrupted parties for Functionality` $\mathbf{InitShareGen}_r$ `using the inputs of the previous step. In addition,` $\mathcal{S}$ `deals with possible aborts is each step. */`

Simulating the preliminary phase of Protocol $\mathbf{MPCUnLtd}_r$:

`/* Protocol` $\mathbf{MPCUnLtd}_r$ `appears in Figure 6. */`

This part of the simulations begins when $\mathcal{S}$ receives the inputs of the active corrupted parties to $\mathbf{InitShareGenWithAbort}_r$. If at least one of the parties aborts, this step is repeated. Next, $\mathcal{S}$ produces the outputs for the active corrupted parties for Functionality $\mathbf{InitShareGen}_r$ using the inputs of the current step. To do so, $\mathcal{S}$ selects some of the unknown information uniformly at random, i.e.,. the value of $i^\star$ and the inputs of the honest parties. Given these values, $\mathcal{S}$ simply follows the three computation parts of Procedure $\mathbf{PrepareResPCKGsAndMSGs}$. An important point to emphasize is that some of the obtained shares are temporary and will later be opened for the actual values obtained from the trusted party during the interaction rounds using the properties of Shamir's secret-sharing scheme.

Simulating interaction rounds for a set of active parties index by L:

`/* Corresponds to Figure 9. */`

The main part of the simulation is to simulate the messages that the honest parties send to the corrupted parties in each round of interaction. Prior to round $i^\star$, it is an easy mission, as all the values in the system are uniform; thus, a uniform selection of these values is indistinguishable from the real selection. In round $i^\star$, the simulator $\mathcal{S}$ communicates with the trusted party and receives the real output of the functionality. The simulator $\mathcal{S}$ performs a process of updating the information that it has from the preliminary phase. This information includes: (1) The default values for each subset of active corrupted parties, (2) The "resumption packages" for each subset of active corrupted parties, and (3) The messages that the corrupted parties have to receive from the honest parties. As we described before, $\mathcal{S}$ is able to perform this process using to the properties of Shamir's secret-sharing scheme.

Simulating the preliminary phase of Protocol $\mathbf{MPCUnLtdAfterAbort}_r$:

`/* Protocol` $\mathbf{MPCUnLtdAfterAbort}_r$ `appears in Figure 11. */`

The simulation of this part is similar to the simulation of the preliminary phase of Protocol $\mathbf{MPCUnLtd}_r$. However, there are two main differences:

1. In the first step, the simulator $\mathcal{S}$ receives a set of shares of the "resumption packages" on behalf of the the corrupted parties and not their inputs.
2. The simulator produces the outputs of the active corrupted parties for $\mathbf{InitShareGen}_r$:
   - If the $i^\star$ round has occurred, then $\mathcal{S}$ already got the real output σ from the trusted party. Thus, $\mathcal{S}$ produces the output of Procedure $\mathbf{PrepareResPCKGsAndMSGs}$ based on this value.
   - If the $i^\star$ round has not occurred yet, $\mathcal{S}$ produces the output of Procedure $\mathbf{PrepareResPCKGsAndMSGs}$ uniformly at random. As before, $\mathcal{S}$ is able to perform this process by the properties of Shamir's secret-sharing scheme, although the set of shares of the corrupted parties is already given.

Figure 12: Sketch of the Simulator.

Getting the inputs from $\mathcal{A}$:

(a) The simulator $\mathcal{S}$ receives a set of inputs $\{x_\ell : \ell \in B \cap L\}$ that the adversary $\mathcal{A}$ submits to $\mathbf{InitShareGenWithAbort}_r$.

Dealing with possible aborts of $\mathcal{A}$:

(b) If a party p_j for $j \in B \cap L$ does not submit an input, i.e., $\mathcal{A}$ sends an "`abort`$_j$" message on behalf of at least one party p_j; then,

i. For each such aborted party p_j, the simulator $\mathcal{S}$ notifies all corrupted parties that p_j aborted and updates $L = L \setminus \{j\}$.

ii. Goto Step (3).

Simulating the computation of default values:

(c) $\mathcal{S}$ prepares outputs for the corrupted parties for Functionality $\mathbf{InitShareGen}_r$:

`/* The inputs of the active corrupted parties are already known to` $\mathcal{S}$`, while the other inputs are unknown. Therefore, uniformly selected inputs are used to build the outputs of the functionality. Later on, they are updated with the correct value. */`

i. The simulator $\mathcal{S}$ selects $i^\star \in \{1, \ldots, r\}$ with uniform distribution.

ii. For each $\ell \in B \cap L$ set $\widehat{z}_\ell = x_\ell$, for each $j \in \{1, \ldots, m\} \setminus (B \cap L)$ the simulator $\mathcal{S}$ selects uniformly at random $\widehat{z}_j \in X_n$, and sets $\sigma \leftarrow f_n(\widehat{z}_1, \ldots, \widehat{z}_m)$.

iii. $\mathcal{S}$ follows the three computation parts of Procedure $\mathbf{PrepareResPCKGsAndMSGs}$ with the value of $i^\star$, the set of inputs $\{\widehat{z}_j\}_{1 \le j \le m}$, and the prescribed output of the protocol σ computed in Step (2(c)i) and Step (2(c)ii).

iv. For each party p_ℓ s.t. $\ell \in B \cap L$, the simulator $\mathcal{S}$ sends to $\mathcal{A}$:

- The messages $M_{1,\ell}, \ldots, M_{r,\ell}$ that p_ℓ broadcasts during the protocol.
- The shares $[s_{1,\ell}]_\ell, \ldots, [s_{r,\ell}]_\ell$.

Dealing with possible aborts of $\mathcal{A}$:

(d) If $\mathcal{A}$ sends an "`abort`$_j$" for some party p_j s.t. $j \in B \cap L$ to $\mathcal{S}$, then, for each such aborted party p_j, the simulator $\mathcal{S}$ updates $L = L \setminus \{j\}$.

3. If an abort of at least one party has occurred, then,

- If $|L| = 1$,
  (a) The simulator $\mathcal{S}$ sends "`abort`$_j$" for each $j \in B \setminus L$ to the trusted party computing $\mathcal{F}$ and receives σ.
  (b) The simulator $\mathcal{S}$ outputs the sequence of messages exchanged between $\mathcal{S}$ and the adversary $\mathcal{A}$ and halts.
- Else, goto Step (2).

4. Else, if no abort has occurred, set $\mathbf{gotOutputFromTP} = \mathit{false}$ and goto "Simulating interaction rounds for a set of active parties index by L".

`/* The boolean variable` **gotOutputFromTP** `is used to ensure that the` $\mathcal{S}$ `simulator communicates with the trusted party at most once, as the model requires. */`

Simulating interaction rounds for a set of active parties index by L:

`/* Corresponds to the part of the protocol appears in Figure 9. */`

To simulate round i for $i = 1, \ldots, r$, the simulator $\mathcal{S}$ proceeds as follows:

/* In each round, the simulator $\mathcal{S}$ sends messages on behalf of the honest parties to the adversary. In round $i^\star$, the simulator $\mathcal{S}$ communicates with the trusted party and receives the real output of the protocol. */

1. If $i = i^\star$ and $\mathbf{gotOutputFromTP} = \mathit{false}$,
   (a) The simulator $\mathcal{S}$ sends the set of inputs $\{x_\ell : \ell \in B \cap L\}$ and "abort$_j$" for each $j \in B \setminus L$ to the trusted party computing $\mathcal{F}$ and receives σ.
   (b) $\mathbf{gotOutputFromTP} = \mathit{true}$.
   /* As a result of obtaining the real output σ from the trusted party, $\mathcal{S}$ has to update some of the information that was computed in advance for the rounds starting from round $i^\star$. This information includes:
   i. The default values for each subset of active corrupted parties.
   ii. The ''resumption packages'', which are based on the default values, also for each subset of active corrupted parties.
   iii. The messages the simulator has to receive from the honest parties.
   */
   (c) The simulator $\mathcal{S}$ updates the default values:
   For each $i^\star \le i \le r$ and for each $J \subset B \cap L$ the simulator $\mathcal{S}$ sets $i^\star_{J,i} = 1$ and $w^i_J = \sigma$.
   (d) The simulator $\mathcal{S}$ updates the "resumption packages":
   For each $i^\star \le i \le r$ and for each $J \subset B \cap L$ s.t. $|J| \ge 2$,
      i. The simulator $\mathcal{S}$ constructs $\mathbf{S}_{J,i}$ consisting of:
      (according to the values from the last step)
      - $\{x_\ell\}_{\ell \in J}$ – The inputs of the (future) active parties.
      - $i^\star_{J,i}$ – the special $i^\star$ value for the (future) protocol.
      - w^i_J – the output value for the (future) protocol.
      ii. The simulator $\mathcal{S}$ shares $\mathbf{S}_{J,i}$ in an $|J|$-out-of-$|J|$ secret-sharing scheme for the parties $\{p_\ell : \ell \in J\}$. For each $\ell \in J$, let $[\mathbf{S}_{J,i}]_\ell$ be the updated share of p_ℓ of the secret $\mathbf{S}_{J,i}$.
   (e) The simulator $\mathcal{S}$ computes the messages of the honest parties:
   /* This is done using the fact that given a set of shares that do not define a secret (being held by $\mathcal{S}$), it is possible to compute efficiently the missing shares (for the honest parties). */
      i. For each $i^\star \le i \le r$ and for each $\ell \in B \cap L$, the simulator $\mathcal{S}$ constructs the string $s_{i,\ell}$ that p_ℓ should reconstruct in round i. I.e,
      - The value $w^i_{\{\ell\}}$.
      - The shares $[\mathbf{S}_{J,i}]_\ell$ for each $J \subset B \cap L$ s.t. $|J| \ge 2$ and $\ell \in J$.
      ii. The simulator $\mathcal{S}$ constructs new shares for the honest parties:
      For each $i^\star \le i \le r$ and for each $\ell \in B \cap L$ let $\{\alpha_\ell\}_{\ell \in B \cap L}$ be the shares prepared in the simulation of the preliminary phase for the $|L|$-out-of-$|L|$ secret-sharing scheme that the corrupted parties hold of the secret $s_{i,\ell}$.
      The simulator $\mathcal{S}$ computes $|L| - |B \cap L|$ shares for the honest parties $s_{i,\ell}$, given the shares $\{\alpha_\ell\}_{\ell \in B \cap L}$ that the corrupted parties hold.

iii. The simulator $\mathcal{S}$ computes the messages that the honest parties have to send in round i: For each honest party p_{q_h}, the simulator $\mathcal{S}$ constructs $M_{i,q_h} \leftarrow ([s_{i,\ell}]_{q_h})_{\ell \neq q_h, \ell \in L}$.

2. For each honest party p_{q_h}, the simulator $\mathcal{S}$ sends the message M_{i,q_h} on behalf of the p_{q_h} to $\mathcal{A}$.
3. If $\mathcal{A}$ sends an "$\texttt{abort}_j$" for some party p_j s.t. $j \in B \cap L$ to $\mathcal{S}$, then,
   (a) For each aborted party p_j, the simulator $\mathcal{S}$ updates $L = L \setminus \{j\}$.
   (b) If $|L| = 1$,
      i. The simulator $\mathcal{S}$ sends "$\texttt{abort}_j$" for each $j \in B \setminus L$ to the trusted party computing $\mathcal{F}$ and receives σ.
      ii. The simulator $\mathcal{S}$ outputs the sequence of messages exchanged between $\mathcal{S}$ and the adversary $\mathcal{A}$ and halts.
   (c) Else, goto "Simulating the preliminary phase of Protocol $\mathbf{MPCUnLtdAfterAbort}_r$".

Simulating the end of round r for a set of active parties indexed by L:

1. The simulator $\mathcal{S}$ outputs the sequence of messages exchanged between $\mathcal{S}$ and the adversary $\mathcal{A}$, and halts.

Simulating the preliminary phase of Protocol $\mathbf{MPCUnLtdAfterAbort}_r$:

`/* Protocol` $\mathbf{MPCUnLtdAfterAbort}_r$ `appears in Figure 11. */`

1. The simulator $\mathcal{S}$ simulates the interaction of $\mathcal{A}$ with $\mathbf{ResShareGenWithAbort}_r$ as follows:

   Getting the inputs from $\mathcal{A}$:

   (a) The simulator $\mathcal{S}$ receives a set of shares of the "resumption packages" – $\mathbf{S}_{L,\ell}$: For each $\ell \in B \cap L$, the simulator $\mathcal{S}$ receives the share $[\mathbf{S}_{L,i}]_\ell$ that $\mathcal{A}$ submits to Functionality $\mathbf{ResShareGenWithAbort}_r$.

   Dealing with possible aborts of $\mathcal{A}$:

   (b) If a party p_ℓ for $\ell \in B \cap L$ does not submit an input, i.e., $\mathcal{A}$ sends an "$\texttt{abort}_j$" on behalf of at least one party p_j, then,
      i. For each such aborted party p_j, the simulator $\mathcal{S}$ notifies all corrupted parties that p_j aborted and updates $L = L \setminus \{j\}$.
      ii. The simulation of the interaction of $\mathcal{A}$ with $\mathbf{ResShareGenWithAbort}_r$ is terminated, i.e., Step (2) below is executed.

   Simulating the computation of default values:

   (c) $\mathcal{S}$ prepares outputs for the corrupted parties for $\mathbf{ResShareGenWithAbort}_r$:

   `/* The inputs of the active corrupted parties are already known to` $\mathcal{S}$`, while the other inputs are unknown. Therefore, uniformly selected inputs are used to build the outputs of the functionality. Later on, during round` $i^\star$`, the real output received they are updated. */`

      i. For each $\ell \in B \cap L$, the simulator $\mathcal{S}$ sets $\widehat{z}_\ell = x_\ell$, for each $j \in \{1, \ldots, m\} \setminus (B \cap L)$ the simulator $\mathcal{S}$ selects uniformly at random $\widehat{z}_j \in X_n$.
      ii. If the abort for which Protocol $\mathbf{MPCUnLtdAfterAbort}_r$ is executed occurred prior round $i^\star$, then,

A. The simulator $\mathcal{S}$ selects $i^\star \in \{1,\ldots,r\}$ with uniform distribution and sets $\sigma \leftarrow f_n(\widehat{z}_1,\ldots,\widehat{z}_m)$.
`/* Observe that the values of` $i^\star$ `and` σ `have already been fixed; however,` $\mathcal{S}$ `is unable to see them as the shares of the honest parties are needed. */`

B. $\mathcal{S}$ follows the three computation parts of Procedure $\mathbf{PrepareResPCKGsAndMSGs}$ with the value of $i^\star$, the set of inputs $\{\widehat{z}_j\}_{1\leq j\leq m}$, and the prescribed output of the protocol σ computed above in Step (1(c)iiA) and Step (1(c)i).

iii. Else ($i^\star$ has already occurred), $\mathcal{S}$ follows the three computation parts of Procedure $\mathbf{PrepareResPCKGsAndMSGs}$ with $i^\star = 1$, the set of inputs $\{\widehat{z}_j\}_{1\leq j\leq m}$, and the prescribed output of the protocol σ that was received in Step (1a) from the dealer in the simulation of the interactive rounds.
`/* The set of inputs` $\{\widehat{z}_j\}_{1\leq j\leq m}$ `might not be compatible with the value` σ`. However, the set of inputs` $\{\widehat{z}_j\}_{1\leq j\leq m}$ `does not have an effect, as all the values that are computed in Procedure` $\mathbf{PrepareResPCKGsAndMSGs}$ `are equal to` σ`. */`

iv. For each party p_ℓ s.t. $\ell \in B \cap L$, the simulator $\mathcal{S}$ sends to $\mathcal{A}$:
- The messages $M_{1,\ell},\ldots,M_{r,\ell}$ that p_ℓ broadcasts during the protocol.
- The shares $[s_{1,\ell}]_\ell,\ldots,[s_{r,\ell}]_\ell$.

Dealing with possible aborts of $\mathcal{A}$:

(d) If $\mathcal{A}$ sends an "`abort`$_j$" for some party p_j s.t. $j \in B \cap L$ to $\mathcal{S}$, then, for each such aborted party p_j, the simulator $\mathcal{S}$ updates $L = L \setminus \{j\}$.

2. If the execution of the simulation that appears in Step (1) prematurely terminated, i.e., an abort of at least one party has occurred, then,
   - If $|L| = 1$,
     (a) The simulator $\mathcal{S}$ sends "`abort`$_j$" for each $j \in B \setminus L$ to the trusted party computing $\mathcal{F}$ and receives σ.
     (b) The simulator $\mathcal{S}$ outputs the sequence of messages exchanged between $\mathcal{S}$ and the adversary $\mathcal{A}$ and halts.
   - Else, if $|L| > 1$, then goto Step (1) in "Simulating the preliminary phase of Protocol $\mathbf{MPCUnLtdAfterAbort}_r$".
3. Else, if no abort has occurred, goto "Simulating interaction rounds for a set of active parties index by L".

5.3 Proof of the Correctness for the Simulation for $\mathrm{MPCUnLtd}_r$

We next prove the correctness of the simulation described in Section 5.2. The general structure of the proof is similar to the one presented in Section 4.4.2. Roughly speaking, we consider the same two random variables $\mathbf{REAL} = (V_{\mathbf{REAL}}, C_{\mathbf{REAL}})$ and $\mathbf{IDEAL} = (V_{\mathbf{IDEAL}}, C_{\mathbf{IDEAL}})$ that we considered in Section 4.4.2. The variable $V_{\mathbf{REAL}}$ describes a possible view of $\mathcal{A}$ in the above hybrid model world, and $C_{\mathbf{REAL}}$ describes a possible output of the honest parties in this world. The random variable $V_{\mathbf{IDEAL}}$ describes the output of the simulator in the ideal world described in Section 2.2 and $C_{\mathbf{IDEAL}}$ is the output of the honest parties in this execution. Our goal is to prove that these two random variables – $\mathbf{REAL}$ and $\mathbf{IDEAL}$ – are within statistical distance $O(1/p)$. A more detailed explanation of these variables appears in Section 4.4.2.

Recall that we assumed in the beginning of Section 5.1 that an adversary never (successfully) forges a signature during the execution of the protocol. Therefore, we present the protocol and its simulator without using signatures, which are briefly described in Section 5.1.1. However, the probability that a polynomial-time adversary successfully forges a signature for any message is negligible; therefore, the statistical distance between the random variables $\mathbf{REAL}$ and $\mathbf{IDEAL}$ can only grow additively by a negligible function compared to the statistical distance between these two random variables without such an assumption.

We first provide a proof for deterministic functionalities in Section 5.3.1 and then, in Section 5.3.2 we extend it for randomized functionalities. The corresponding main lemmas are Lemma 5.3 and Lemma 5.10, in which we prove the correctness of the simulation by showing that the statistical distance between the two random variables is $O(1/p)$. The general structure of our proof is showing that the statistical distance between the two random variables is bounded (asymptotically) by the sum of the following two quantities: (1) The probability of guessing $i^\star$ on time in the m-protocol. (2) The probability that at least one party aborts in the m-protocol prior to round $i^\star$, and then, at least one party aborts in the inner protocol and $i^\star_{L,i} = 1$.

In the next claim we show a lower bound on the probability that all the values that the adversary obtains in a round $i < i^\star$ of Protocol $\mathbf{MPCUnLtd}_r$ indicate that it is an $i^\star$-like round, i.e., all the values w^i_J that the adversary sees are equal to a fixed value and all the values $i^\star_{J,i}$ that the adversary sees are all equal to 1.

Claim 5.1. *Let $\mathcal{F}$ be a (possibly randomized) functionality computed by Protocol $\mathbf{MPCUnLtd}_r$ and $d(n)$ be the size of its domain. Fix some inputs $x_1, \ldots, x_m$ and w such that $\Pr[f_n(x_1, \ldots, x_m) = w] \geq \epsilon$ for some $\epsilon > 0$. Then, the probability that in a round $i < i^\star$ all the values w^i_J that the adversary sees are equal to the specific w and all the values of $i^\star_{J,i}$ that the adversary sees are equal to 1 is at least*

$$\frac{1}{\sqrt{r}} \left(\frac{\epsilon}{d(n)^m} \right)^{\Omega(2^m)}.$$

Furthermore, if $\mathcal{F}$ is deterministic (thus, $\Pr[f_n(x_1, \ldots, x_m) = w] = 1$), then, this probability is at least $1/\left(\sqrt{r} \cdot d(n)^{m \cdot \Omega(2^m)}\right)$.

Proof. We start with the deterministic case. Recall that $\widehat{x}_1, \ldots, \widehat{x}_m$ are the inputs used to obtain w^i_J; that is, $w^i_J = f_n(\widehat{x}_1, \ldots, \widehat{x}_m)$, where $\widehat{x}_\ell = x_\ell$ for each $j \in J$ and $\widehat{x}_j$ is selected uniformly at random from X_n for every $j \in \{1, \ldots, m\} \setminus J$. We lower-bound the probability that $w^i_J = w$ by the probability that $\widehat{x}_j = x_j$ for every $j \in \{1, \ldots, m\} \setminus J$. The probability that $\widehat{x}_j = x_j$ for each $j \notin J$ is $1/d$. Therefore, the probability that $w^i_J = w$ for a specific $J \subseteq B$ is at least $(1/d)^{m-|J|} \geq (1/d)^m$. As $|B| \leq m-1$, the adversary sees less than 2^{m-1} values. Thus, the probability that for all $J \subseteq B$, all the values w^i_J are all equal to w is at least $(1/d)^{m \cdot \Omega(2^m)}$.

Next, we bound the probability that in a round $i < i^\star$ all the values of $i^\star_{J,i}$ that the adversary sees are equal to 1. Recall that these values are selected independently, where for every $J \subseteq B$ s.t. $|J| \geq 2$, $\Pr[i^\star_{J,i} = 1] > (1/r)^{1/\left(2^{|L|} - |L|\right)}$. The adversary sees $i^\star_{J,i}$ for every set J of size at least two that is contained in $B \cap L$. As there is at least one honest party, the adversary sees the values of at most $2^{|L|-1} - (|L| - 1) \leq 2^{|L|-1} - |L|/2$ sets. Therefore, the probability that all the values $i^\star_{J,i}$ that the adversary sees are equal to 1 is at least:

$$\begin{aligned} \Pr\left[\bigwedge_{J \subseteq B : |J| \geq 2} i^\star_{J,i} = 1 \right] &= \prod_{J \subseteq B : |J| \geq 2} \Pr[i^\star_{J,i} = 1] \\ &\geq \left((1/r)^{1/\left(2^{|L|} - |L|\right)} \right)^{2^{|L|-1} - |L|/2} = 1/\sqrt{r}. \end{aligned}$$

Therefore, as the values of w_J^i are independent of the values of $i^\star_{J,i}$ when $i < i^\star$, the probability that in a round $i < i^\star$ all the values w_J^i that the adversary sees are equal to a specific w and all the values of $i^\star_{J,i}$ that the adversary sees are equal to 1 is at least $1/\left(\sqrt{r} \cdot d^{m \cdot \Omega(2^m)}\right)$.

For randomized functionality $\mathcal{F}$, the evaluation of $f_n(\widehat{x}_1, \ldots, \widehat{x}_m)$ has two steps: first $\widehat{x}_j$ is randomly chosen from X_n for every $j \notin L$ and then the randomized functionality is evaluated. Thus, we first consider the analysis of the deterministic case and given that in a round $i < i^\star$ all the values w_J^i that the adversary sees are equal to a specific w and all the values of $i^\star_{J,i}$ that the adversary sees are equal to 1, we analyze the probability that all the values w_J^i that the adversary sees are mapped to the same value. We showed above that $\mathcal{A}$ obtains fewer than $2^{m-1} - m$ values w_J^i in each round $i < i^\star$. Therefore, to get the lower bound for the randomized case, we can simply multiply the lower bound of the deterministic case by $\epsilon^{\Omega(2^m)}$. □

5.3.1 Proof for Deterministic Functionalities

Notation 5.2. *We say that a protocol for a set of parties indexed by L has a fake-$i^\star$ if $i^\star_{L,i} = 1$ as a result of the first assignment in Step (1a) in Procedure* $\mathbf{PrepareResPCKGsAndMSGs}$ *(see Figure 8).*

Observe that $i^\star_{L,i}$ might be equal to 1 also as a result of a uniform selection of $i^\star \in \{1, \ldots, r\}$, but we consider only the first case, in which $i^\star_{L,i} = 1$ with probability $r^{-1/\left(2^{|L|} - |L|\right)}$.

In the next lemma, we prove the correctness of the simulation for deterministic functionalities by using Claim 5.1.

Lemma 5.3. *Let $\mathcal{F}$ be a deterministic functionality, let $\mathcal{A}$ be a non-uniform polynomial-time adversary corrupting at most $m - 1$ parties in an execution of Protocol* $\mathbf{MPCUnLtd}_r$*, and let $\mathcal{S}$ be the simulator described in Section 5.2 (where $\mathcal{S}$ controls the same parties as $\mathcal{A}$). Then, for every $n \in \mathbb{N}$, for every $\vec{y} \in (X_n)^m$, and for every* $\mathbf{aux} \in \{0,1\}^*$*,*

$$\mathbf{SD}\left(\mathbf{REAL}_{\mathbf{MPCUnLtd}_r, \mathcal{A}(\mathbf{aux})}(\vec{y}, 1^n), \mathbf{IDEAL}_{\mathcal{F}, \mathcal{S}(\mathbf{aux})(\vec{y}, 1^n)}\right) = \frac{d^{m \cdot O(2^m)}}{\sqrt{r(n)}},$$

where $d = d(n)$ is the size of the domain of $\mathcal{F}$, and $r(n)$ is the number of rounds in the protocol.

Proof. Our goal here is to show that the statistical distance between the above two random variables is at most as stated in Lemma 5.3. In our proof, we will use the following pairs of random variables, which describe partial executions of the protocol and the corresponding simulation until a certain point of the entire execution. We assume that each of the following random variables is concatenated by a random variable that represents the value of $i^\star$ in the execution. We do not specify the random variable explicitly in the following description to simplify the notations. When we analyze a specific execution, we can assume that the value of $i^\star$ is known.

1. • $\mathbf{REAL}^{\mathbf{Init}}$ and $\mathbf{IDEAL}^{\mathbf{Init}}$:

   Random variables corresponding to execution of Protocol $\mathbf{MPCUnLtd}_r$ until the end of the preliminary phase. The random variable $\mathbf{REAL}^{\mathbf{Init}}$ contains the messages that the adversary exchanged with Functionality $\mathbf{InitShareGen}_r$. Similarly, the random variable $\mathbf{IDEAL}^{\mathbf{Init}}$ describes the simulation of $\mathbf{InitShareGenWithAbort}_r$ and it contains the messages obtained by $\mathcal{S}$.

2. • $\mathbf{REAL}^i$ and $\mathbf{IDEAL}^i$ for $1 \leq i \leq r$.

Random variables corresponding to the execution until the end of the i-th round of Protocol $\mathbf{MPCUnLtd}_r$. For each $1 \leq i \leq r-1$, the random variable $\mathbf{REAL}^i$ contains the concatenation of $\mathbf{REAL}^{\mathbf{Init}}$ that was described above, with the messages that the adversary receives from the honest parties until the end of round i. Similarly, $\mathbf{IDEAL}^i$ contains a concatenation of $\mathbf{IDEAL}^{\mathbf{Init}}$ with the messages obtained by the simulator $\mathcal{S}$ until the end of round i. Observe that if round number r ends properly, i.e., no abort has occurred during the entire execution of the protocol, then the honest parties get the output, and the random variables $\mathbf{REAL}^r$ and $\mathbf{IDEAL}^r$ also contain the output of the honest parties.

3. • $\mathbf{REAL}^{\mathbf{Res}_i}$ and $\mathbf{IDEAL}^{\mathbf{Res}_i}$ for $1 \leq i \leq r$.

   Random variables describing the execution until the end of the preliminary phase of the *first* execution of Protocol $\mathbf{MPCUnLtdAfterAbort}_r$ that was executed as a result of an abort in a round i. Observe that Protocol $\mathbf{MPCUnLtdAfterAbort}_r$ might be executed several times; however, we focus only on the first execution of this protocol. The random variable $\mathbf{REAL}^{\mathbf{Res}_i}$ contains a concatenation of $\mathbf{REAL}^i$, where i is the round in which the abort has occurred, with the messages that the adversary exchanged with Functionality $\mathbf{ResShareGen}_r$. Similarly, $\mathbf{IDEAL}^{\mathbf{Res}_i}$ contains a concatenation of $\mathbf{IDEAL}^i$, where i is the round in which the abort has occurred, with the simulation of the first execution of $\mathbf{ResShareGenWithAbort}_r$ and it contains the messages that were exchanged between the adversary and the simulator $\mathcal{S}$.

The proof consists of the following informal claims:

1. $\mathbf{REAL}^{\mathbf{Init}}$ and $\mathbf{IDEAL}^{\mathbf{Init}}$ are identically distributed as all the information obtained by the adversary in the execution of $\mathbf{InitShareGen}_r$ and $\mathbf{InitShareGenWithAbort}_r$ are shares, which are totally uniform and independent strings.

2. If no abort has occurred, then, for every $1 \leq i \leq r$ both random variables $\mathbf{REAL}^i$ and $\mathbf{IDEAL}^i$ are distributed identically. Prior to $i^\star$, the views in both worlds are independent of the prescribed output. While, after $i^\star$, the views in both worlds are consistent with the prescribed output. In addition, the adversary's decision to abort before $i^\star$ is independent of the output. If during the entire execution no abort has occurred, then, the random variables $\mathbf{IDEAL}^r$ and $\mathbf{REAL}^r$ are distributed identically, as $\mathbf{REAL}^{r-1}$ and $\mathbf{IDEAL}^{r-1}$ are distributed identically and in the last round both views are consistent with the final output, which appears in both random variables.

3. If an abort has occurred, the parties execute Protocol $\mathbf{MPCUnLtdAfterAbort}_r$. Similarly to the explanation of why $\mathbf{REAL}^{\mathbf{Init}}$ and $\mathbf{IDEAL}^{\mathbf{Init}}$ are distributed identically, all the information obtained by the adversary during the execution of $\mathbf{ResShareGen}_r$ and $\mathbf{ResShareGenWithAbort}_r$ are shares, which are uniform and independent strings. Therefore, for each $1 \leq i \leq r$, the statistical distance between the random variables $\mathbf{REAL}^{\mathbf{Res}_i}$ and $\mathbf{IDEAL}^{\mathbf{Res}_i}$ is equal to the statistical distance between $\mathbf{REAL}^i$ and $\mathbf{IDEAL}^i$.

4. If an abort of at least one party occurs in round i of Protocol $\mathbf{MPCUnLtd}_r$, then, 3 scenarios are possible:

   (a) $i = i^\star$**.** In such a case, a constant statistical distance is possible between the two random variables $\mathbf{REAL}$ and $\mathbf{IDEAL}$ as in the ideal world the parties already received the output, while in the real world the parties are asked to execute a protocol that is, informally, independent of the previous view. However, we use Lemma 2.6 to show that the probably of such scenario is low, as it is hard to guess the right value of $i^\star$.

(b) $i < i^\star$. In such a case, both random variables $\mathbf{REAL}^{\mathbf{Res}_i}$ and $\mathbf{IDEAL}^{\mathbf{Res}_i}$ are distributed identically. However, the execution of the protocol continues, i.e., the active parties execute a protocol, in which the adversary gets "another chance" to cause a statistical distance. We analyze this scenario conditioned on whether the sub-protocol has a fake-$i^\star$ or not. If the sub-protocol has a fake-$i^\star$, then the adversary can abort in the first round and cause a constant statistical distance; however, the probably of such a scenario is low. If the sub-protocol does not have a fake-$i^\star$, then, the views in both worlds are independent of the prescribed output and the inputs of the honest parties. Therefore, the executed sub-protocol is independent of all the past seen values and can be seen as a "new protocol"; thus, the resulting statistical distance can be added to the original one. By that, we upper-bound asymptotically the statistical distance in the m-party protocol.

(c) $i > i^\star$. This case is equivalent to the case in which no additional abort has occurred as the views in both worlds are both consistent with the prescribed output.

A detailed explanation follows:

The Initialization Step:

Claim 5.4. $\mathbf{SD}\left(\mathbf{REAL}^{\mathbf{Init}}, \mathbf{IDEAL}^{\mathbf{Init}}\right) = 0$.

Proof. The information obtained as a result of executing $\mathbf{InitShareGen}_r$ and $\mathbf{InitShareGenWithAbort}_r$ are shares of the parties in $L \cap B$ in an $|L|$-out-of-$|L|$ Shamir secret-sharing scheme, since there is at least one honest party in L, the shares that the adversary sees in both cases are uniformly distributed. □

An Honest Interaction Step of the Protocol (no Aborts):

Claim 5.5. *For each $i < i^\star$, if no abort occurred until the end of round i, then* $\mathbf{SD}\left(\mathbf{REAL}^i, \mathbf{IDEAL}^i\right) = 0$.

Proof. We argue that both in the real protocol and in the simulation, the view of $\mathcal{A}$ is identically distributed in the two worlds. In the ideal world, $\mathcal{S}$ produces a view that is random as there is no value received from the trusted party. In the real world, the random process is similar and the interaction with $\mathcal{A}$ is based on values that are outputs of evaluations of the function f_n on the same input distributions. The adversary does not learn anything about the inputs of the honest parties; hence, its decision to abort does not depend on any new information it obtains during the interaction rounds so far. □

Claim 5.6. *For each $i \geq i^\star$, if no abort occurred until the end of round i, then* $\mathbf{SD}\left(\mathbf{REAL}^i, \mathbf{IDEAL}^i\right) = 0$.

Proof. In this case, the view must contain the prescribed output of the protocol σ. In the real-world execution, this value is equal to the output value of all sets for any round $i > i^\star$ (recall that the output value of the honest parties is determined by one such value), and in the simulation it equals to the value obtained from the trusted party. Thus, in both scenarios, the view must be consistent with $i^\star$ and with the output value; hence, the view completely determines the output. Since the simulator $\mathcal{S}$ follows the same random process in interacting with $\mathcal{A}$ as in the real-world execution and the output is chosen according to the same distribution, the probabilities are the same. Recall that according to the explanation in Section 2.5, the simulator can compute efficiently the shares of the adversary in order to obtain the right values in each round. □

Premature abort During the Interaction Rounds: We next consider the cases when an abort occurs. When an abort of at least one party occurs during the execution of Protocol $\mathbf{MPCUnLtd}_r$, the remaining parties compute Functionality $\mathbf{ResShareGen}_r$ in a secure-with-abort with cheat-detection model, in which each active party p_ℓ inputs the corresponding shares of the "resumption packages" – $\mathbf{S}_{J,\ell}$ for each $J \subseteq L$ s.t. $\ell \in J$ and the default value $w^0_{\{\ell\}}$. Each active party gets back as an output a set of messages and a set of private shares for exacting r rounds of interaction. We next prove several claims on the statistical distance between the random variables in the lemma.

Claim 5.7. $\mathbf{SD}\big(\mathbf{REAL}^{\mathbf{Res}_i}, \mathbf{IDEAL}^{\mathbf{Res}_i}\big) = \mathbf{SD}\big(\mathbf{REAL}^{i}, \mathbf{IDEAL}^{i}\big)$ *for each* $1 \leq i \leq r$.

Proof. Similarly to Claim 5.4, the information obtained as a result of executing $\mathbf{ResShareGen}_r$ and $\mathbf{ResShareGenWithAbort}_r$ are shares, which are uniformly distributed. Therefore, the statistical distance between $\mathbf{REAL}^i$ and $\mathbf{IDEAL}^i$ is the same as the statistical distance between $\mathbf{REAL}^{\mathbf{Res}_i}$ and $\mathbf{IDEAL}^{\mathbf{Res}_i}$ as we claim. □

For the next claims we need the following notation that explicitly presents the number of active parties.

Notation 5.8. *We use* $\mathbf{IDEAL}_j$ *and* $\mathbf{REAL}_j$ *to refer to the random variables* $\mathbf{IDEAL}$ *and* $\mathbf{REAL}$*, respectively, where* j *is the number of active parties in the protocol.*

Claim 5.9. *If an abort has occurred in* $i < i^\star$*, then*

$$\mathbf{SD}\Big(\mathbf{REAL}_m, \mathbf{IDEAL}_m\Big) \leq \mathbf{SD}\Big(\mathbf{REAL}_{m-1}, \mathbf{IDEAL}_{m-1}\Big) + r^{-1/(2^m - m)}.$$

Proof. If an abort has occurred in $i < i^\star$, then the remaining parties, indexed by L, execute the $|L|$-party protocol using the "resumption packages". In this protocol the adversary gets another chance to cause a statistical distance. We consider two cases: (1) The sub-protocol has a fake-$i^\star$ and (2) The sub-protocol does not have a fake-$i^\star$. In the first case, the adversary can abort in the first round and cause a constant statistical distance, similarly to aborting in round $i^\star$ in the original protocol. However, the probability that a sub-protocol has a fake-$i^\star$ is at most $r^{-1/(2^m - m)}$. In the second case, the views in both worlds are independent of the prescribed output and the inputs of the honest parties; thus, the executed sub-protocol is independent of all the past. Therefore, in this case, the statistical distance between the two random variables is upper-bounded by statistical distance between the random variables in an $(m-1)$-party protocol. □

Claims 5.4–5.7 and Claim 5.9 describe the cases when the statistical distance appears. We next combine them and upper-bound the statistical distance between $\mathbf{IDEAL}_m$ and $\mathbf{REAL}_m$.

$$\begin{aligned}
&\mathbf{SD}(\mathbf{IDEAL}_m, \mathbf{REAL}_m) \\
&\leq \mathbf{Pr}[\text{Aborting prior to round } i^\star]\Big(\mathbf{SD}\Big(\mathbf{REAL}_{m-1}, \mathbf{IDEAL}_{m-1}\Big) + r^{-1/(2^m - m)}\Big) \\
&\quad + \mathbf{Pr}[\text{Aborting in round } i^\star] \cdot \mathbf{1} \\
&\quad + \mathbf{Pr}[\text{Aborting after round } i^\star] \cdot \mathbf{0}. \qquad (3)
\end{aligned}$$

By Claim 5.1 for deterministic functionalities, the probability of every view of the adversary occurring in a round $i < i^\star$ is at least $1/\left(\sqrt{r} \cdot d(n)^{m \cdot \Omega(2^m)}\right)$. By Lemma 2.6 with $\beta = 0$, we get that

$$\mathbf{Pr}[\text{Aborting in round } i^\star] \leq \frac{\sqrt{r} d(n)^{m \cdot O(2^m)}}{r} = \frac{d(n)^{m \cdot O(2^m)}}{\sqrt{r}}. \qquad (4)$$

By using (4) and the fact that $\mathbf{Pr}[\text{Aborting prior round } i^\star] \leq 1$, we can rewrite (3) as follows:

$$\mathbf{SD}(\mathbf{IDEAL}_m, \mathbf{REAL}_m) \leq \mathbf{SD}(\mathbf{IDEAL}_{m-1}, \mathbf{REAL}_{m-1}) + r^{-1/(2^m-m)} + \frac{d(n)^{m \cdot O(2^m)}}{\sqrt{r}}.$$

Therefore,

$$\begin{aligned}
&\mathbf{SD}(\mathbf{IDEAL}_m, \mathbf{REAL}_m) \\
&\leq \sum_{\ell=2}^{m} \left(r^{-1/(2^\ell-\ell)} + \frac{d(n)^{\ell \cdot O(2^\ell)}}{\sqrt{r}} \right) \\
&\leq m \cdot \max_{2 \leq \ell \leq m} \left\{ r^{-1/(2^\ell-\ell)} + \frac{d(n)^{\ell \cdot O(2^\ell)}}{\sqrt{r}} \right\} \\
&\leq m \cdot \left(\max_{2 \leq \ell \leq m} \left\{ r^{-1/(2^\ell-\ell)} \right\} + \max_{2 \leq \ell \leq m} \left\{ \frac{d(n)^{\ell \cdot O(2^\ell)}}{\sqrt{r}} \right\} \right) \\
&\leq m \cdot \left(r^{-1/2} + \frac{d(n)^{m \cdot O(2^m)}}{\sqrt{r}} \right). \qquad (5)
\end{aligned}$$

Finally, by assuming that $d(n) > 1$, we conclude that

$$\mathbf{SD}(\mathbf{IDEAL}_m, \mathbf{REAL}_m) = \frac{d^{m \cdot O(2^m)}}{\sqrt{r}}.$$

□

5.3.2 Proof for Randomized Functionalities

We next describe the changes that should be made in the claims and their proofs of the deterministic case in order to obtain a proof for the randomized case. That is, the main difference is the analysis of the probability to abort in round $i^\star$. Formally, we next prove the following lemma:

Lemma 5.10. *Let $\mathcal{F}$ be a (possibly randomized) functionality, let $\mathcal{A}$ be a non-uniform polynomial-time adversary corrupting at most $m-1$ parties indexed by B in an execution of Protocol $\mathbf{MPCUnLtd}_r$, and let $\mathcal{S}$ be the simulator described in Section 5.2 (where $\mathcal{S}$ controls the same parties as $\mathcal{A}$). Then, for every $n \in \mathbb{N}$, for every $\vec{y} \in (X_n)^m$, and for every $\mathbf{aux} \in \{0,1\}^*$,*

$$\mathbf{SD}(\mathbf{IDEAL}_m, \mathbf{REAL}_m) = \frac{g(n) \cdot d(n)^{O(m)}}{r^{2^{-m}}},$$

where $d(n)$ and $g(n)$ are the sizes of the range and the domain of $\mathcal{F}$, respectively, and r is the number of rounds in the protocol.

Proof. We next use Lemma 2.6 and Claim 5.1 to bound the probability that the adversary guesses $i^\star$. Let $x_1, \ldots, x_m$ be the inputs obtained by the dealer in the preprocessing phase of Protocol $\mathbf{MPCWithDLtd}_r$. Let p_0 be a parameter specified below. We call an output value w *heavy* if $\mathbf{Pr}[w = f_n(x_1, \ldots, x_m)] > 1/(p_0 \cdot g)$, otherwise, we call w *light* (where the probability is taken over the randomness in computing the functionality). Let S_h and S_ℓ be a partition to heavy and light values, respectively.

Observe that since there are at most g possible values of $f_n(x_1, \ldots, x_m)$, the probability that w is light, by the union bound, is at most $1/p_0$. Next, by Claim 4.2 where $\epsilon = 1/(p_0 \cdot g)$, the probability that all the values w_L^i that the adversary sees in round $i < i^\star$ are equal to w for a heavy w, and the values of $i^\star_{J,i}$ that the adversary sees are equal to 1 is at least $(1/\sqrt{r})\,(1/(p_0 \cdot g \cdot d^m))^{\Omega(2^m)}$. By Lemma 2.6 with $\beta = 1/p_0$ and $\alpha = (1/\sqrt{r})\,(1/(p_0 \cdot g \cdot d^m))^{\Omega(2^m)}$, the probability that the adversary guesses $i^\star$ it is at most

$$\frac{1}{p_0} + \frac{\left((1/\sqrt{r})\,(1/(p_0 \cdot g \cdot d^m))^{\Omega(2^m)}\right)^{-1}}{r} = \frac{1}{p_0} + \frac{(p_0 \cdot g \cdot d^m)^{O(2^m)}}{\sqrt{r}}.$$

We take $p_0 = r^{2^{-m}}/(g \cdot d^m)$ and obtain that the total probability that the adversary guesses $i^\star$ is at most

$$\frac{1}{p_0} + \frac{(p_0 \cdot g \cdot d^m)^{O(2^m)}}{\sqrt{r}} = \frac{g \cdot d^{O(m)}}{r^{2^{-m}}}.$$

Therefore, by using Equation (3) from the deterministic case and applying a similar analysis to (5) we obtain the proof for the lemma. □

5.4 A $1/p$-Secure Protocol for Polynomial Range

Protocol $\mathbf{MPCUnLtd}_r$, described in Section 5.1, is $1/p$-secure when either the functionality is deterministic and the size of the domain is polynomial, or when the functionality is randomized and both the domain and range of the functionality are polynomial. In this section, we present a modification of the protocol presented in Section 5.1 achieving a $1/p$-secure protocol for (possibly randomized) functionalities whose size of the range is polynomial (even if the size of the domain of the functionality is not polynomial). The modification is similar to the modification in Section 4.3. Only one modification should be made, and it is in the way that each value w_J^i is chosen prior to round $i^\star$. The main idea is to cause each one of the rounds prior round $i^\star$ "to look-like" round $i^\star$. To achieve this goal, with probability $1/(2p)$ the values w_J^i are chosen as a random value in the range of f_n and with probability $1 - 1/(2p)$, the values are chosen exactly as in the original protocol described in Figure 8. Formally, in Procedure $\mathbf{PrepareResPCKGsAndMSGs}$ described in Figure 8, we replace the first step in the "Computing default values" phase with the following step:

Computing default values:

1. For each $1 \leq i < i^\star$ and for each $J \subset L$,
   (a) With probability $r^{-1/(2^{|L|}-|L|)}$, set $i^\star_{J,i} = 1$.
   With the remaining probability, select $i^\star_{J,i} \in \{1, \ldots, r\}$ with uniform distribution.
   (b)
   - With probability $1/(2p)$, select uniformly at random $z_L^i \in Z_n$ and set $w_J^i = z_L^i$.
   - With the remaining probability $1 - 1/(2p)$, for each $\ell \in J$ set $\widehat{z}_\ell = x_\ell$, for each $j \in \{1, \ldots, m\} \setminus J$ select uniformly at random $\widehat{z}_j \in X_n$, and set $w_J^i \leftarrow f_n(\widehat{z}_1, \ldots, \widehat{z}_m)$.

Denote the resulting protocol by $\mathbf{MPCUnLtdPolyRange}_r$. As described before, this modification causes that the probability that all values held by the adversary are equal prior to round $i^\star$ is higher; thus, the probability that the adversary guesses $i^\star$ is smaller. However, in the modified protocol, the honest parties can output a value that is not possible given their inputs, and, in general, we cannot simulate the case (which happens with probability $1/(2p)$) when the output is chosen with uniform distribution from the range. As the latter happens with probability $1/(2p)$, we can ignore this case in the simulation and conclude that the protocol is $1/p$-secure.

5.5 Proof of Security for Protocol $\mathrm{MPCUnLtdPolyRange}_r$

Lemma 5.11. *Let $\mathcal{F}$ be a (possibly randomized) functionality. For every non-uniform polynomial-time adversary $\mathcal{A}$ corrupting at most $m-1$ parties in an execution of Protocol $\mathbf{MPCUnLtdPolyRange}_r$, there exists a simulator $\mathcal{S}_T$ in the ideal model that simulates the execution of $\mathcal{A}$ (where $\mathcal{S}_T$ controls the same parties as $\mathcal{A}$), such that for every $n \in \mathbb{N}$, for every $\vec{y} \in (X_n)^m$, and for every $\mathbf{aux} \in \{0,1\}^*$*

$$\mathbf{SD}(\mathbf{IDEAL}_m, \mathbf{REAL}_m) \leq \frac{(2p(n) \cdot g(n))^{2^m}}{\sqrt{r(n)}} + \frac{1}{2p(n)},$$

where $g(n)$ is the size of the range of $\mathcal{F}$.

Proof. The simulators and their proofs for Protocol $\mathbf{MPCUnLtd}_r$ and the protocol discussed in this section - Protocol $\mathbf{MPCUnLtdPolyRange}_r$, are similar; we only (informally) present the differences between the two simulators and the two proofs.

The modified simulator. Recall that the only difference between Protocol $\mathbf{MPCUnLtd}_r$ and Protocol $\mathbf{MPCUnLtdPolyRange}_r$ is the first step in the "Computing default values" phase in Procedure $\mathbf{PrepareResPCKGsAndMSGs}$ described in Figure 8. In $\mathbf{MPCUnLtdPolyRange}_r$, each value w_L^i prior to round $i^\star$ is chosen with probability $1/(2p)$ as a random value from the range of f_n and with probability $1-1/(2p)$ it is chosen just like in $\mathbf{MPCUnLtd}_r$. There are two modifications to the simulator. The first modification in the simulator is in part that "$\mathcal{S}$ follows the three computation parts of Procedure $\mathbf{PrepareResPCKGsAndMSGs}$" that appear in Step (1(c)iiB) and in Step (2(c)iii). These instructions describe the computation of w_L^i for $i < i^\star$. The modified simulation step appears below.

1. For each $1 \leq i < i^\star$ and for each $J \subset L$,
   (a) With probability $r^{-1/\left(2^{|L|}-|L|\right)}$, set $i^\star_{J,i} = 1$.
   With the remaining probability, select $i^\star_{J,i} \in \{1,\ldots,r\}$ with uniform distribution.
   (b)
   - With probability $1/(2p)$, select uniformly at random $z_J^i \in Z_n$ and set $w_J^i = z_J^i$.
   - With the remaining probability $1-1/(2p)$, for each $\ell \in J$ set $\widehat{z}_\ell = x_\ell$, for each $j \in \{1,\ldots,m\} \setminus J$ select uniformly at random $\widehat{z}_j \in X_n$, and set $w_J^i \leftarrow f_n(\widehat{z}_1,\ldots,\widehat{z}_m)$.

The second modification is less obvious. Recall that both random variables appearing in the lemma contain the output of the honest parties. In the ideal world, the honest parties always output f_n applied to their inputs. In the real world, if an abort occurs in round $i < i^\star$, with probability $1/(2p)$, the honest parties output a random value from the range of f_n. It is hard to simulate the output of the honest parties in first case.[5] We simply modify the simulator such that with probability $1/(2p)$ the simulator returns $\perp$, i.e., it announces that the simulation has failed. The formal description of the additional step appears below.

- If at least one active party aborts in round $i < i^\star$,
  - With probability $1/(2p)$,
    1. Send the set of inputs $\{x_\ell : \ell \in B \cap L\}$ and "$\mathtt{abort}_j$" for each $j \in B \setminus L$ to the trusted party computing $\mathcal{F}$ (and receive σ).
    2. Return $\perp$ and stop the simulation.
  - With the remaining probability $1-1/(2p)$, proceed with the original simulation.

[5] For example, there might not be possible inputs of the corrupted parties causing the honest parties to output such output.

The modified proof. The proof to the simulator for $\mathbf{MPCUnLtdPolyRange}_r$ remains basically the same, except for two changes. We first modify Claim 5.1 and prove a slightly different claim, which changes the probability of the adversary guessing $i^\star$.

Claim 5.12. *Let $g(n)$ be the size of the range of the (possibly randomized) functionality $\mathcal{F}$ computed by Protocol $\mathbf{MPCUnLtdPolyRange}_r$ and $w \in Z_n$. Then, the probability that in a round $i < i^\star$ all the values w_J^i that the adversary sees are equal to a specific w and all the values of $i^\star_{J,i}$ that the adversary sees are equal to 1 is at least equal to w is at least*

$$\frac{1}{\sqrt{r}}\left(\frac{1}{2p(n)\cdot g(n)}\right)^{2^m}.$$

Proof. According to the protocol, there are two different ways to produce each value w_J^i in round $i < i^\star$: (1) Compute f_n on a set of inputs and a set of uniformly selected values from the domain of the functionality, and (2) Set w_J^i as a uniformly selected value from the range of the functionality. We ignore the first case. In the second case, with probability $1/2p$, the value w_J^i is uniformly selected from the range. Hence, the probability that w_J^i is equal to a specific value is at least $1/(2p \cdot g)$.

As in the proof of Claim 5.1, in each round of the protocol, $\mathcal{A}$ obtains less than $2^{m-1} - m \leq 2^m$ values. Therefore, we conclude that the probability that all the values w_J^i that $\mathcal{A}$ obtains in round $i < i^\star$ are all equal to w is at least $(1/(2p \cdot g))^{2^m}$. In addition, all the values of $i^\star_{J,i}$ that the adversary sees have to be equal to 1; as in Claim 5.1, the probability for that is at least $1/\sqrt{r}$. By multiplying the two probabilities, we obtain the claim. □

By applying the Lemma 2.6 we conclude that the probability of the adversary guessing $i^\star$ correctly in Protocol $\mathbf{MPCUnLtdPolyRange}_r$ is at most $(2p \cdot g)^{2^m}/\sqrt{r}$. In the case of an abort of at least one party in round $i < i^\star$, with probability $1 - 1/(2p)$ in both the ideal world and real world, the value that the honest parties output is the evaluation of f_n on the inputs of the active parties and random inputs for the parties that aborted. However, with probability $1/(2p)$, if an abort of at least one party occurs prior to round $i^\star$, the output of the honest parties Protocol $\mathbf{MPCUnLtdPolyRange}_r$ is a random value from the range of f_n; the simulator fails to simulate the execution in this case and outputs $\perp$. Thus,

$$\begin{aligned}
&\mathbf{SD(IDEAL, REAL)}\\
&\leq \mathbf{Pr}[\text{Aborting in round } i^\star] + (1/2p)\cdot \mathbf{Pr}[\text{Aborting prior round } i^\star]\\
&\leq \frac{(2p(n)\cdot g(n))^{2^m}}{\sqrt{r(n)}} + \frac{1}{2p(n)}.
\end{aligned}$$

Therefore, the statistical distance is as claimed. □

6 Best of Both Worlds – The $1/p$ Way

We study the question of whether or not it is possible to construct "best of both worlds" protocols, when the fall-back security guarantee is $1/p$-security or $1/p$-security-with-abort, which is, informally, $1/p$-security enhanced with ability (of the adversary) to prematurely abort the computation. We investigate whether protocols with these weaker forms of security are possible when full privacy cannot be guaranteed. In this section we describe our feasibility results.

In Section 6.1 we constructed protocols that guarantee full security whenever fewer than half of the parties are corrupted, and $1/p$-security-with-abort otherwise. In Section 6.2 we design protocols that guarantee full security whenever fewer than half of the parties are corrupted, and $1/p$-security otherwise. For both cases, we show that such protocols exist for every functionality $\mathcal{F}$ with a constant number of parties when both domain and range are polynomial in the security parameter. Clearly, any protocol guaranteeing the requirements of Section 6.2 (i.e., has fall-back $1/p$-(full)-security) also guarantees the requirements of Section 6.1. Nevertheless, the protocols presented in Section 6.1 are simpler and more efficient than the protocols that we present in Section 6.2. We next formally state the results of this section.

Theorem 5. *Let $\mathcal{F}$ be an m-party (possibly randomized) functionality. If enhanced trap-door permutations exist, and if m is constant and the size of the domain $d(n)$ and the size of the range $g(n)$ are bounded by a polynomial in the security parameter n, then for any polynomial $p(n)$ there is an $r(n)$-round $1/p(n)$-secure protocol computing $\mathcal{F}$ tolerating up to $m-1$ corrupted parties and, in addition, guarantees full security in the presence of an honest majority, where $r(n) = p(n)^2 \cdot (g(n) \cdot d(n)^m)^{O(2^m)}$. Furthermore, if $\mathcal{F}$ is deterministic, we let $r(n) = p(n)^2 \cdot d(n)^{O(m2^m)}$, and obtain an $r(n)$-round protocol for computing $\mathcal{F}$, with the same properties and under the same assumptions.*

Theorem 6. *Let $\mathcal{F}$ be an m-party (possibly randomized) functionality where the size of the domain is $d(n)$ and the size of the range is $g(n)$, and let $p(n)$ be any polynomial. Let $t \in \mathbb{N}$, be such that $m/2 \leq t < 2m/3$ and $r(n) = p(n) \cdot (2 \cdot d(n)^m \cdot g(n) \cdot p(n))^{2^t}$. If $r(n)$ is bounded by a polynomial in n and enhanced trap-door permutations exist, then there is an $r(n)$-round m-party $1/p$-secure protocol computing $\mathcal{F}$, tolerating up to t corrupted parties, and in addition guarantees full security in the presence of an honest majority. Furthermore, if $\mathcal{F}$ is deterministic, we let $r(n) = p(n) \cdot d(n)^{m \cdot 2^t}$, and obtain an $r(n)$-round protocol for computing $\mathcal{F}$, with the same properties and under the same assumptions.*

6.1 Best of Both Worlds – The $1/p$-Security-With-Abort Variant

In this section we consider protocols obtaining a best of both worlds type property, with a fall-back security guarantee for the case of no honest majority being $1/\mathrm{poly}$-security-with-abort. That is, we consider protocols that guarantee full security whenever fewer than half of the parties are corrupted, and $1/\mathrm{poly}$-security-with-abort otherwise.

We first define the notion of $1/\mathrm{poly}$-security-with-abort more formally. We say that a protocol is $1/\mathrm{poly}$-secure-with-abort if, similarly to Definition 2.2, the ideal and the real world are computationally $1/p$-indistinguishable; however, the computation in the ideal world follows the steps of the ideal world of security-with-abort and cheat-detection (see Section 2.3) with the minor difference that all the "$\texttt{cheat}_j$" messages are replaced with the $\perp$ sign, indicating that an abort of at least one party occurred without specifying the identities of any of the aborted parties.

For the sake of being more general, we allow an upper bound t on the number of parties for which the second guarantee should hold. That is, we allow protocols that for some polynomial $p(\cdot)$ and some $t \in \mathbb{N}$ provide the following security properties:

1. If the adversary corrupts fewer than half of the parties, then full security is guaranteed.
2. If the adversary corrupts t' of the parties, where $t' \leq t$ then $1/p$-security-*with-abort* is guaranteed, for some polynomial $p(\cdot)$.

The previously best known protocol of the best of both worlds type, by Katz [27], had fall-back security of $1/p$-security-with-abort. The protocol of [27] works for functionalities with up to exponential size domain

and range, and a polynomial number of parties. Our protocols work when the domain is of polynomial size and the number of parties is constant. However, the fall-back security guarantee of the protocol of [27] relies on an assumption that the adversary is non-rushing, which, in many cases is unrealistic. Our protocol is secure even if the adversary is rushing.

We first describe a $\mathbf{3}$-party protocol achieving the above security requirements (we later sketch how to generalize it to an m-party protocol). As in all our protocols, we describe a two-phased protocol. The first phase is a preliminary phase in which the parties compute a given functionality (securely-with-abort with cheat-detection). The outputs of this functionality include the messages that each party broadcasts throughout the protocol execution, together with some masking, so that each party can extract some information (known only to it) from the publicly broadcast messages of the other parties. In the second phase, the parties interact with each other (via a broadcast channel), using the messages they have been instructed to send by the output of the preliminary phase. For simplicity of presentation, we assume that in the second phase the adversary is in the *augmented fail-stop*[6] model. That is, all parties follow the protocol with two exceptions: (i) the corrupted parties are able to use different inputs than the inputs specified to them, and (ii) the corrupted parties can abort the computation at any time. This is without loss of generality, since we have already demonstrated how to limit the adversary to aborts by signing any message that could ever be sent in the protocol. Using this assumption, we can omit the discussion regarding the signing of the messages. In addition, we first describe the protocol assuming that all the parties participate honestly in preliminary phase.

Let us first give a verbal overview of the $\mathbf{3}$-party protocol that computes a functionality $\mathcal{F}$. Consider three parties p_1, p_2, p_3 holding inputs y_1, y_2, y_3 (respectively). In the preliminary phase, the parties engage in a secure-with-abort with cheat-detection protocol, in which output values $\sigma^i_{\{a,b\}}$ are assigned to each pair of parties (p_a, p_b) for each round $1 \leq i \leq r$ in the protocol. Corrupted parties are able to change their inputs to this functionality; hence, we denote the actual inputs used in this computation by x_1, x_2, x_3 (specifically, an honest party p_j will use $x_j = y_j$ as its input). The values of the outputs of the preliminary phase are selected similarly to our previous protocols. I.e., the output $w \leftarrow \mathcal{F}(x_1, x_2, x_3)$ is (secretly) computed and a special round $1 \leq i^\star \leq r$ is (secretly) selected uniformly. The output value of each pair (p_a, p_b) and for each round $1 \leq i < i^\star$ is computed by evaluating $\mathcal{F}$ on x_a, x_b and a randomly selected input $\widehat{x}_c$ (for party p_c, such that, $c \neq a, b$); $\widehat{x}_a = x_a$ and $\widehat{x}_b = x_b$ and compute $\sigma^i_{\{a,b\}} \leftarrow \mathcal{F}(\widehat{x}_1, \widehat{x}_2, \widehat{x}_3)$. The output value of each pair (p_a, p_b) and for each round $i^\star \leq i \leq r$ is set to the prescribed output of the protocol, i.e., $\sigma^i_{\{a,b\}} \leftarrow w$.

The outputs of the preliminary phase are a two-layered secret sharing of the above information. The outer layer is a $\mathbf{3}$-out-of-$\mathbf{3}$ Shamir secret-sharing scheme of the shares of the inner layer. The inner layer consists of secret shares for each round i and each pair (p_a, p_b) in a $\mathbf{2}$-out-of-$\mathbf{2}$ Shamir secret-sharing scheme, where the shares are given to parties p_a, p_b.

In each round of interaction $1 \leq i \leq r$ the parties reconstruct the outer secret-sharing layer, allowing each party to learn its shares in the inner secret-sharing scheme for each pair to which it belongs (for round i). We consider two cases:

- If a party p_c prematurely aborts in round i, then the remaining two parties p_a, p_b broadcast their shares in the inner secret-sharing scheme to reconstruct $\sigma^{i-1}_{\{a,b\}}$ (if $i = 1$, they compute it using a secure-with-abort protocol). If one of the parties aborts during the reconstruction, then the remaining party outputs $\perp$. In addition, if two parties abort prematurely during one of the rounds of the protocol,

[6] We borrow our terminology from [17] who discussed a model of augmented semi-honest adversaries in which the adversary is semi-honest, except that it is allowed to change its input to the computation.

then the remaining party outputs $\perp$.

- If the last round was successfully completed, then all parties broadcast all their shares in the inner secret sharing of $\sigma^r_{\{a,b\}}$. If an honest party cannot reconstruct any secret $\sigma^r_{\{a,b\}}$, then it outputs $\perp$.

To verify that the above protocols satisfy the security requirements, we need to consider two cases:

There is an honest majority. In this case, at most one party is corrupted. Thus, the adversary learns nothing before aborting (if a party aborts before the end of round r). This is true, since a single party can learn nothing about the secrets of the inner $\mathbf{2}$-out-of-$\mathbf{2}$ secret-sharing scheme of each round i. Furthermore, these shares are never reconstructed before the end of round r, unless the corrupted party aborts (before the reconstruction). In addition, the remaining (honest) two parties can always reconstruct the secret of the inner secret-sharing scheme, and, hence, output a correct value.

There is no honest majority. In this case, there are two corrupted parties. We first analyze the case where the two corrupted parties do not abort in the same round. We claim that the adversary can only cause problems (for the simulator) by aborting in round i^*. However, the probability of the adversary guessing i^* is limited by the size of the domain times the number of rounds in the protocol. The exact analysis is almost identical to the one appearing in the two-party protocol of [24] (for polynomial size domain). This is because the adversary here has a very similar view to an adversary in the two-party protocol of [24], since in each round of the protocol it learns a single output of the functionality (which before round $i^\star$ is independent of the honest party's input, and from $i^\star$ and on is fixed). The probability of guessing the value of $i^\star$ is $\mathbf{1}/p$, and thus, the probability of causing constant computation distance is bounded by $\mathbf{1}/p$, therefore, in this case the protocol is "only" $\mathbf{1}/p$-secure. In the case where the two corrupted parties abort in the same round, the honest party outputs $\perp$. The worst scenario is when one party, say p_3, aborts in round $i^\star$ and another party, say p_2, does not send its share of round $i^\star - \mathbf{1}$. In this case, the adversary learns two values of the function, that is, $\mathcal{F}(x_1, x_2, x_3)$ and $\mathcal{F}(\widehat{x}_1, x_2, x_3)$, while the honest party learns no output and outputs $\perp$. However, as the probability of guessing $i^\star$ is at post $\mathbf{1}/p$, the resulting protocol is $\mathbf{1}/p$-secure-with-abort.

Dealing with Premature Aborts During the Preliminary Phase. Recall that in the description above we do not consider the case where one or two of the parties abort during (or before) the preliminary phase. If only one party aborts, then the remaining two parties simply execute a constant round secure-with-abort protocol for computing $\mathcal{F}$ (directly), e.g., the protocols of [30, 4], where the protocol selects uniformly at random an input for the third party. If an additional abort occurs during this protocol, then the remaining party outputs $\perp$. In the other case, where two parties abort during the preliminary phase, the remaining party outputs $\perp$. To verify that security requirements hold in the scenario of premature termination during the preliminary phase, we observe that in the case of honest majority, the remaining two parties are honest; thus, any protocol that keeps the privacy of the inputs will suffice. In the other case, if there is no honest majority, outputting $\perp$ meets the requirements of the security definition.

We next describe the generalization of the $\mathbf{3}$-party protocol described above to an m-party protocol. The basic structure of the m-protocol remains the same as in $\mathbf{3}$-party protocol, but the following changes should be made.

1. **Assign values to sets of size $\lfloor m/2+1 \rfloor$.** The protocol in the preliminary phase assigns a value σ_J^i for each set of parties indexed by J where $|J| = \lfloor m/2 \rfloor + 1$. For each round $1 \leq i < i^\star$, each value σ_J^i is computed by evaluating $\mathcal{F}$ on the input x_j of p_j for each $j \in J$ and a randomly selected input $\widehat{x}_\ell$ for each $\ell \notin J$. For each round $i^\star \leq i \leq r$, each value σ_J^i is equal to the evaluation of $\mathcal{F}$ on the set of all the inputs of the parties.

2. **Different thresholds.** The inner layer is a $(\lfloor m/2 \rfloor + 1)$-out-of-$(\lfloor m/2 \rfloor + 1)$ Shamir secret-sharing scheme, while the outer layer is a $(\lfloor m/2 \rfloor + 2)$-out-of-$m$ Shamir secret-sharing scheme.

3. **Keep reconstructing the outer secret-sharing layer as long as possible.** In each round of interaction $1 \leq i \leq r$, the parties try to reconstruct the outer secret-sharing layer. If there are enough valid shares, i.e., there are at least $\lfloor m/2 \rfloor + 2$ active parties that broadcast valid shares, then the parties move to the next round. If there are $\lfloor m/2 \rfloor + 1$ active parties, the set of active parties reconstructs their corresponding value from the last round. If there are less that $\lfloor m/2 \rfloor + 1$ active parties or if not all active parties send their shares in the reconstruction, then all active parties output $\perp$. In any case, the active parties keep a list of the aborted parties and once a party gets into this list, it is treated as an aborted party through the entire protocol.

4. **Dealing with aborts in the preliminary phase.** If fewer than $m - (\lfloor m/2 \rfloor + 2) + 1 = \lceil m/2 \rceil - 1$ parties abort during (or before) the preliminary phase, i.e., there are enough active parties to reconstruct the outer secret-sharing scheme, the functionality of the protocol in the first phase selects inputs for the aborted parties uniformly at random and continues with the prescribed instructions. Else, if exactly $\lceil m/2 \rceil + 1$ active parties abort the preliminary phase, the remaining parties execute a secure-with-abort protocol for computing the $\mathcal{F}$ (directly), where the protocol selects uniformly at random inputs for the aborted parties. If an additional abort occurs during this protocol or more than $\lceil m/2 \rceil + 1$ active parties abort in the preliminary phase, then the remaining parties output $\perp$.

6.2 Best of Both Worlds – The $1/p$-(full)-Security Variant

In this section we show how to transform $1/p$-secure protocols of a certain type into protocols that retain the same security for the case of no honest majority, while guaranteeing full security whenever fewer than half of the parties executing the protocol are corrupted. Intuitively, the transformation works if the original protocol has full security against a weaker adversary that can only abort at the beginning of each round (i.e., before seeing the messages of the honest parties for this round); that is, we require that in each round either everyone sends a message or no one does. Specifically, this transformation can be applied to all protocols in this paper that have full correctness. Note that protocols that do not have full correctness (at least for the case of an honest majority) do not guarantee full security for the case of an honest majority.

The basic structure of protocols that can be transformed. For simplicity of presentation we first present our transformation considering protocols with a certain structure. Later we argue that if the original protocol satisfies the structural requirements, and, in addition, satisfies some definition of security (i.e., is $1/\operatorname{poly}$-secure and the adversary must use the rushing property to gain advantage), then the resulting protocol has full security when a majority of the parties are honest (while preserving the original security guarantees when only a minority of the parties are honest). Consider an m-party protocol for computing a functionality $\mathcal{F}$ that has the following structure:

1. The interaction starts with a preliminary phase in which the parties execute a secure-with-abort with cheat-detection protocol for computing the messages that the parties are to send in the next r interaction rounds; after this phase, each party p_j holds a (signed) message M_j^i for each round $1 \leq i \leq r$.

2. In each interaction round i each party p_j broadcasts the message M_j^i.

3. Any failure of party p_j to broadcast the signed message as prescribed by the protocol is considered as an abort of p_j. The adversary can cause the protocol to prematurely terminate by instructing some $t_A < \lceil \frac{m}{2} \rceil$ corrupted parties to abort. Unless premature termination takes place, the protocol proceeds normally (that is, as long as fewer than t_A of parties have aborted).

   In the case of premature termination, the remaining parties engage in a protocol Π_{TERM} for agreeing on the output of the protocol, based on the view of the parties in the protocol so far. More specifically, the decision on the output is based on the outputs of the (remaining) parties from the preliminary phase, on the messages broadcast until round $i-1$, and on the set of parties that have aborted.

The transformation. We present a general transformation for protocols having the above structure. The core of the change is a mask we add to the messages of the parties in each round. This mask is shared in a $(\lfloor \frac{m}{2} \rfloor + 1)$-out-of-$m$ secret-sharing scheme. Hence, the messages of the parties are disclosed in the original protocol if and only if a majority of the parties work together to reconstruct the appropriate masks. Below we explain this change in more detail.

1. We change the output of the preliminary phase by adding a mask to the output of the computation, which is shared in an $(\lfloor \frac{m}{2} \rfloor + 1)$-out-of-$m$ Shamir secret-sharing scheme. More specifically, for each message M_j^i (for party p_j and round i) we select a random string r_j^i, and we let $\hat{M}_j^i = M_j^i \oplus r_j^i$ be the message that party p_j is instructed to broadcast in round i. In addition, each party also receives a share of r_j^i in a $(\lfloor \frac{m}{2} \rfloor + 1)$-out-of-$m$ Shamir secret-sharing scheme. The message $\hat{M}_j^i$ and the shares of its mask r_j^i are all signed.

2. Each interaction round of the original protocol is executed in a two-phased round. In the first phase, each party p_j broadcasts the message $\hat{M}_j^i$. In the second phase, the parties reconstruct all masks of round i by broadcasting all shares of masks r_j^i, for $1 \leq j \leq m$.

3. Any failure of party p_j to broadcast the signed message as prescribed by the protocol is considered as an abort of p_j (including messages added by the transformation).

   The adversary can cause the protocol to prematurely terminate by instructing at least t_A (where $t_A < \lceil \frac{m}{2} \rceil$) corrupted parties to abort. Unless premature termination takes place, the protocol proceeds normally. We consider three possible cases for premature termination in round i, depending on whether premature termination occurs in the first or second phase of round i and on whether the reconstruction of the masks (in the second phase) is successful or not:

   Premature termination in first phase: The remaining parties behave as if the original protocol was terminated at the beginning of round i. That is, they engage in a protocol Π_{TERM} for agreeing on the output of the protocol, based on the messages broadcast until round $i-1$ and on the set of parties that have aborted D.

   Premature termination in second phase – unsuccessful mask reconstruction: The remaining parties behave as if the original protocol was terminated at the beginning of round i. Intuitively,

since the masks are not reconstructed the parties do not obtain the full view of the respective execution of the original protocol until round i, and hence agree on the output as implied by the view until round $i-1$ (as in the former case).

Premature termination in second phase – successful mask reconstruction: The remaining parties behave as if the original protocol was terminated at the beginning of round $i+1$. Intuitively, since the masks are reconstructed, the parties obtained the full view of the respective execution of the original protocol until round i. Hence, they engage in a protocol $\mathbf{\Pi_{TERM}}$ for agreeing on the output of the protocol, based on the messages broadcast until round i (and on the set of parties that have aborted D).

The security requirements on the original protocol. Consider protocols that in addition to having the above structure, also satisfy the following requirements:

1. The outputs of the preliminary phase obtained by corrupted parties reveal nothing to the adversary about the output of the protocol, and, thus, while the adversary can cause this phase to fail, this does not affect the security of the whole protocol. E.g., in our protocols this is obtained by the output of the preprocessing phase being shared among the parties in a $(t+1)$-out-of-m secret-sharing scheme, where t is an upper bound on the number of malicious parties.

2. The premature termination protocol $\mathbf{\Pi_{TERM}}$ is fully secure whenever it is executed with an honest majority.

3. If the adversary is limited to terminating the protocol at the beginning of each interaction round (before seeing the messages that honest parties send in this round), then the security of the whole protocol is reduced to the security of the premature termination protocol $\mathbf{\Pi_{TERM}}$.

4. The protocol is $1/p$-secure against a malicious adversary that can corrupt up to t parties.

Lemma 6.1. *For every protocol that has the structure defined above and, in addition, satisfies the latter four requirements, the protocol resulting from the transformation is (i) fully secure against a malicious adversary that can corrupt any strict minority of the parties, and (ii) $1/p$-secure against a malicious adversary that can corrupt up to t parties.*

Proof. We next consider two cases, the first case is when the adversary corrupts a strict minority of the parties, and the second case is that at least half of the parties are malicious.

In the case of an honest majority, the shares of r_j^i that the corrupted parties see do not reveal anything to the adversary as long as the shares of honest parties are not revealed (these shares are only revealed in the second phase of round i). Thus, if the adversary causes a premature termination during the first phase of round i, then it has no more information than is obtained in the original protocol (by an adversary corrupting the same subset of parties) until the beginning of round i. By the property of the original protocol, such an adversary can cause as much harm as it can in the premature termination protocol (given the joint input of the parties, i.e., the communication thus far). However, since an honest majority is guaranteed, this sub-protocol is fully secure, and hence, also the full protocol is fully secure in this case. Notice that in the case of an honest majority, if the first phase succeeds then the second phase succeeds.

In the case of no honest majority, it is quite straightforward to argue that an adversary attacking the transformed protocol is not more powerful than an equivalent adversary for the original protocol. Intuitively, this is true since once the adversary sees the messages of the corrupted parties, the masks add no information

and are simply random strings. To formalize this intuition, we consider the original protocol to be our ideal functionality, and show that the transformed protocol emulates the original one. That is, we show that the view of the adversary $\mathcal{A}$ in the new protocol can be simulated by its view in the original protocol when corrupting the same subset of parties. Roughly speaking, this is done by the simulator, upon receiving the messages of the corrupted parties, selecting masks for the messages of the corrupted parties and shares of these masks. Then, the simulator sends to the adversary the masked messages and the shares of the masks for the corrupted parties (all computed as prescribed in our transformation). In an interaction phase, if the adversary $\mathcal{A}$ prematurely aborts in round i, either in the first phase or in the second phase but without allowing the reconstruction of the masks to go through, then the simulator aborts in the middle of round i. If the adversary allows the reconstruction of the masks in the second phase of round i and causes premature termination after that, then the simulator aborts at the beginning of round $i+1$. □

Applying the transformation to the premature termination protocol $\Pi_{\mathbf{TERM}}$. In the above description of the transformation we required the protocol $\Pi_{\mathbf{TERM}}$ to be fully secure with an honest majority (recall that $\Pi_{\mathbf{TERM}}$ is the protocol that the parties engage in when premature termination occurs). However, it is also possible to consider a protocol $\Pi_{\mathbf{TERM}}$ that is not fully secure to begin with, but only $1/\operatorname{poly}$-secure. Nevertheless, if by applying the same transformation to this sub-protocol (which is executed with strictly fewer parties than the main protocol), it can be turned into a new sub-protocol with the desired properties. As a concrete example, consider Protocol $\mathbf{MPCUnLtd}_r$ being executed with $m = 5$ parties. In this case, an honest majority means that at least 3 parties are honest. Consider the case that p_1 and p_2 are corrupted. In the case that p_1 aborts in round i, the remaining four parties $p_2, \ldots, p_5$ execute a 4-party protocol. If they use the original 4-party protocol, then p_2 can still bias the output of the protocol with some noticeable probability. However, if this protocol is also transformed, then it is fully secure when at most one party is corrupted, yielding the overall protocol fully-secure when at most two parties are corrupted. In the general case, we apply Lemma 6.1 on Theorem 1 and Theorem 3, and obtain Theorem 5 and Theorem 6, respectively.

7 Impossibility Results

7.1 Impossibility of $1/p$-secure Computation with Non-Constant Number of Parties

For deterministic functions, our protocols are efficient when the number of parties m is constant and the size of the domain or range is at most polynomial (in the security parameter n) or when the number of parties is $\log\log n$ and the size of the domain is constant. We next show that, in general, there is no efficient protocol when the number of parties is $m(n) = \omega(1)$ and the size of the domain is polynomial, and when $m(n) = \omega(\log n)$ and the size of the domain of each party is 2. This is done using the following impossibility result of Gordon and Katz [24].

Theorem 7 ([24])**.** *For every $\ell(n) = \omega(\log n)$, there exists a deterministic 2-party functionality $\mathcal{F}$ with domain and range $\{0,1\}^{\ell(n)}$ that cannot be $1/p$-securely computed for $p \geq 2 + 1/\operatorname{poly}(n)$.*

We next state and prove our impossibility results.

Theorem 8. *For every $m(n) = \omega(\log n)$, there exists a deterministic $m(n)$-party functionality $\mathcal{F}'$ with domain $\{0,1\}$ that cannot be $1/p$-securely computed for $p \geq 2 + 1/\operatorname{poly}(n)$ without an honest majority.*

Proof. Let $\ell(n) = m(n)/2$ (for simplicity, assume $m(n)$ is even). Let $\mathcal{F} = \{f_n\}_{n\in\mathbb{N}}$ be the functionality guaranteed in Theorem 7 for $\ell(n)$. The function f_n has two inputs, each input is $\ell(n)$-bit long; that is, the number of bits in the inputs of f_n is $2\ell(n) = m(n)$. Define an $m(n)$-party deterministic functionality $\mathcal{F}' = \{f'_n\}_{n\in\mathbb{N}}$, where in f'_n party p_j gets the jth bit of the inputs of f_n and the outputs of f_n and f'_n are equal. Assume that $\mathcal{F}'$ can be $1/p$-securely computed by a protocol Π' assuming that $t(n) = m(n)/2$ parties can be corrupted. This implies a $1/p$-secure protocol Π for $\mathcal{F}$ with two parties, where the first party simulates the first $t(n)$ parties in Π' and the second party simulates the last $t(n)$ parties. The $1/p$-security of Π is implied by the fact that any adversary $\mathcal{A}$ for the protocol Π can be transformed into an adversary $\mathcal{A}'$ for Π' controlling $m(n)/2 = t(n)$ parties; as $\mathcal{A}'$ cannot violate the $1/p$-security of Π', the adversary $\mathcal{A}$ cannot violate the $1/p$-security of Π. □

Theorem 9. *For every $m(n) = \omega(1)$, there exists a deterministic $m(n)$-party functionality $\mathcal{F}''$ with domain $\{0,1\}^{\log n}$ that cannot be $1/p$-securely computed for $p \geq 2 + 1/\operatorname{poly}(n)$ without an honest majority.*

Proof. Let $\ell(n) = 0.5m(n)\log n$ and let $\mathcal{F} = \{f_n\}_{n\in\mathbb{N}}$ be the functionality guaranteed in Theorem 7 for $\ell(m)$. We divide the $2\ell(n)$ bits of the inputs of f_n into $m(n)$ blocks of length $\log n$. Define an $m(n)$-party deterministic functionality $\mathcal{F}'' = \{f''_n\}_{n\in\mathbb{N}}$, where in f''_n party p_j gets the jth block of the inputs of f_n and the outputs of f_n and f''_n are equal. As in the proof of Theorem 8, a $1/p$-secure protocol for $\mathcal{F}''$ implies a $1/p$-secure protocol for $\mathcal{F}$ contradicting Theorem 7. □

The above impossibility results should be contrasted with the coin-tossing protocol of [5], which is an efficient $1/p$-secure protocol even when m is polynomial in the security parameter and the number of bad parties is $m(n)/2 + O(1)$. Notice that in both our impossibility results the size of the range is super-polynomial (as we consider the model where all parties get the same output). It is open if there is an efficient $1/p$-secure protocol when the number of parties is not constant and the size of both the domain and range is polynomial.

7.2 Impossibility of Achieving "The Best of Both Worlds" for General Functionalities

Above we showed that $1/p$-secure computation is impossible in general when the number of parties is $m(n) = \omega(1)$ and the size of the domain is polynomial, and when $m(n) = \omega(\log n)$ and the size of the domain of each party is 2. Since a "Best of Both Worlds" type protocol with fall-back $1/p$-security is in particular $1/p$-secure, the same impossibility results are implied for protocols of this type (i.e., guaranteeing full security with an honest majority and $1/p$-security otherwise). We show that such protocols are impossible in general, even when allowing the fall-back security to be the weaker notion of $1/p$-security-with-abort. Hence, we show that the results discussed in Section 6 are somewhat optimal.

We start by showing in Section 7.2.1 that for general functionalities (i.e., where both domains and both ranges may be super-polynomial), it is impossible to construct even 3-party protocols that simultaneously achieve full security for the case of an honest majority (i.e., at most one corrupted party) and $1/p$-security-with-abort with no honest majority. We then use this result in Section 7.2.2 to prove general impossibility results.

In [26, 27], an impossibility result was given for the case that the fall-back security is security-with-abort (as opposed to $1/p$-security-with-abort). By combining ideas from [26, 27] and from Section 7.1 (and hence from [24]), we are able to show impossibility results even with the weaker notion of $1/p$-security-with-abort as the fall-back security.

7.2.1 Impossibility of Achieving "The Best of Both Worlds" for a 3-Party Functionality

In this section we prove that it is not possible to construct best of both worlds type protocols for general functionalities (with super-polynomial size domain and range). Specifically, we prove the following lemma:

Lemma 7.1. *For every $\ell(n) = \omega(\log n)$, there exists a deterministic functionality $\mathcal{F}$ for three parties p_1, p_2, p_3, such that:*

- *The inputs domain of p_1, p_3 and the range are $\{0,1\}^{O(\ell(n))}$.*
- *p_2 has no input.*
- *$\mathcal{F}$ cannot be efficiently computed simultaneously guaranteeing full security against a single fail-stop party and $1/p(\cdot)$-security-with-abort against two fail-stop parties, for any $p > 2 + \frac{1}{\text{poly}}$.*

Proof. Our proof uses a variant of the functionality used in the impossibility result of [24] for obtaining $1/p$-secure protocols for general 2-party functionalities. We define a deterministic 3-party function

$$\mathbf{Swap}_n^3 : \{0,1\}^{O(\ell(n))} \times \emptyset \times \{0,1\}^{O(\ell(n))} \to \{0,1\}^{2\ell(n)} .$$

We define the function relative to a fixed length function $\ell(n) = \omega(\log n)$ and an information theoretic $\mathbf{MAC}$ scheme $(\mathbf{Gen}, \mathbf{Mac}, \mathbf{Ver})$ for messages of length $2 \cdot \ell(n)$ with key length $O(\ell(n))$ and tag length $\ell(n)$. We define $\mathbf{Swap}^3 = \{\mathbf{Swap}_n^3\}_{n \in \mathbb{N}}$ as follows:

$$\mathbf{Swap}_n^3((x_1, \mathbf{tag}_1, \mathbf{key}_3, \mathbf{tag}'_3, \mathbf{key}'_3), \lambda, (x_3, \mathbf{tag}_3, \mathbf{key}_1, \mathbf{tag}'_1, \mathbf{key}'_1))$$
$$\stackrel{\text{def}}{=} \begin{cases} (x_1, x_3) & \text{if } \mathbf{Ver}_{\mathbf{key}_1}(x_1, \mathbf{tag}_1) = 1 \text{ and } \mathbf{Ver}_{\mathbf{key}_3}(x_3, \mathbf{tag}_3) = 1, \\ * & \text{otherwise.} \end{cases}$$

In the above, λ denotes the empty string and $*$ is a symbol that is not in the input domain. That is, if p_1 and p_3 each hold a message x_i and an authentication of this message $\mathbf{tag}_i$ (authenticated with the key held by the other party), then the messages are revealed to all parties. The functionality ignores the two rightmost elements in p_1's input and in p_3's input. These elements will be used by the proof to derive a security failure.

We next prove that there is no 3-party protocol for computing $\mathbf{Swap}^3$ with a polynomial number of rounds that simultaneously guarantees full security against adversaries that corrupt a single party and $1/p(\cdot)$-security-with-abort against adversaries that corrupt two parties, for any polynomial $p > 2 + \frac{1}{\text{poly}}$. The idea of our proof is that whenever the adversary limits its deviation from the protocol to instructing at most one party to abort, then the protocol cannot distinguish the case where the adversary corrupt two parties from the case where it corrupts only a single party. Hence, the protocol can never instructs the parties to output $\perp$ in such executions. By the definition of $1/p$-security-with-abort, this in effect means that we are back to the scenario of $1/p$-(full)-security. Furthermore, by considering adversaries that corrupt either p_1 and p_2, or p_2 and p_3 (neither of which ever instruct p_2 to abort), the adversary has the view of a strict majority of the parties and we are essentially back to the case of a dishonest majority aiming at $1/p$-security, and we can specifically apply the arguments of [24] for the two-party scenario. The formal proof is given below.

We fix the security parameter n and consider an ideal evaluation of $\mathbf{Swap}_n^3$ in which both p_1 and p_3 receive inputs such that the output of the functionality on these inputs is of the form (x_1, x_3) (i.e., they bear proper tags). In addition, each party also receives a pair $(\mathbf{tag}', \mathbf{key}')$, where $\mathbf{tag}'$ is an authentication

tag for the other party's input x_i, and $\mathbf{key'}$ is the key with which this authentication was obtained. We will later use the fact that this pair allows each party to check whether a given value x is the first element in the other party's input (this will be used in our real-world attack). Formally, we consider ideal world executions, where:

- x_1, x_3 are each selected uniformly at random from $\{0,1\}^{2\cdot\ell(n)}$.
- $\mathbf{key}_1, \mathbf{key}'_1, \mathbf{key}_3, \mathbf{key}'_3$ are selected by applying algorithm $\mathbf{Gen}$ to 1^n.
- $\mathbf{tag}_1 \leftarrow \mathbf{Mac}_{\mathbf{key}_1}(x_1), \mathbf{tag}'_1 \leftarrow \mathbf{Mac}_{\mathbf{key}'_1}(x_1), \mathbf{tag}_3 \leftarrow \mathbf{Mac}_{\mathbf{key}_3}(x_3), \mathbf{tag}'_3 \leftarrow \mathbf{Mac}_{\mathbf{key}'_3}(x_3)$.
- p_1's input is $(x_1, \mathbf{tag}_1, \mathbf{key}_3, \mathbf{tag}'_3, \mathbf{key}'_3)$.
- p_3's input is $(x_3, \mathbf{tag}_3, \mathbf{key}_1, \mathbf{tag}'_1, \mathbf{key}'_1)$.

We consider two adversaries, $\mathcal{A}$ and $\mathcal{B}$. The adversary $\mathcal{A}$ corrupts p_1 and p_2 and instructs them to execute the protocol faithfully until, at some point, it instructs p_1 to abort. The adversary $\mathcal{B}$ corrupts p_2 and p_3 and instructs them to execute the protocol faithfully until, at some point, it instructs p_3 to abort. Neither $\mathcal{A}$ nor $\mathcal{B}$ instructs p_2 to abort. We say that $\mathcal{A}$ wins if $\mathcal{A}$ learns x_3 while p_3 fails to output x_1. We say that $\mathcal{B}$ wins if $\mathcal{B}$ learns x_1 while p_1 fails to output x_3.

Winning in the ideal-world. In the ideal-world, no adversary can win with more than a negligible probability. To see this, consider for example an adversary $\mathcal{A}_{\mathbf{IDEAL}}$ that corrupts p_1 and p_2. In order to win, $\mathcal{A}_{\mathbf{IDEAL}}$ can try two possible strategies:

1. $\mathcal{A}_{\mathbf{IDEAL}}$ may try to replace its input to the functionality so that the functionality returns a pair (x'_1, x_3) with $x'_1 \neq x_1$ (specifically, the functionality does not return $*$ as output). To do so, $\mathcal{A}_{\mathbf{IDEAL}}$ must find (and use as input to the functionality) a pair $(x'_1, \mathbf{tag}''_1)$, such that $x'_1 \neq x_1$ and $\mathbf{Ver}_{\mathbf{key}_1}(x'_1, \mathbf{tag}''_1) = 1$. However, by the properties of the one-time $\mathbf{MAC}$, this is impossible except with negligible probability.
2. Otherwise, $\mathcal{A}_{\mathbf{IDEAL}}$ may try to guess x_3 based on $(x_1, \mathbf{tag}_1, \mathbf{key}_3, \mathbf{tag}'_3, \mathbf{key}'_3)$, and use some junk input causing the output of the functionality to be $*$. One might think that this is possible since p_1 holds an authentication of x_3. However, since x_3 is a uniform $2\ell(n)$-long string and $\mathbf{tag}'_3$ is of length only $\ell(n)$, then even given the key $\mathbf{key}'_3$ there is information theoretically negligible probability of guessing x_3 correctly (for simplicity, assume the $\mathbf{MAC}$ scheme is regular, i.e., there are $2^{\ell(n)}$ strings x that are mapped to the tag $\mathbf{tag}'_3$ by $\mathbf{key}'_3$).

Note that if $\mathcal{A}_{\mathbf{IDEAL}}$ sends $x_1, \mathbf{tag}_1, \mathbf{key}_3$ as the first three parameters of p_1's input, then it does not win, since p_3 learns x_1.

Executions in the real-world where at most one party aborts. Our adversaries (which we describe in more detail below) are fail-stop (i.e., the only way they deviate from the prescribed protocol is by instructing a party to abort) and furthermore, they only instruct at most one party to abort. By the full security guarantee against a single corrupted (specifically, fail-stop) party, this implies that the output of the two remaining parties is $\perp$ only with negligible probability (since in the ideal-world with an honest majority, the parties never get output $\perp$). For simplicity, we assume that this happens with zero probability. When describing our adversaries, we will consider a mental game in which the adversary simulates the case that the honest

party has aborted in round i and computes the output of the two remaining (actually corrupted) parties in such an event. The above argument explains that this computed output is never $\perp$, that is, it is either $*$ or of the form (x, x').

By the full security guarantee when all parties behave honestly, we also have that if no party ever aborts the protocol, then the output of the execution is correct, except with negligible probability. For simplicity we assume that if no party ever aborts, then we have perfect correctness. Hence, below we only consider executions in which the output is never $\perp$ (even if one party aborts), and with full correctness (when none of the parties abort).

Winning in the real-world. We show that in the real-world there is an adversary that wins with probability almost $\mathbf{1/2}$. Roughly speaking, this adversary simulates in each round the event that, say, p_3 aborts and tries to learn the x_3; if the adversary learns the output it aborts, and otherwise proceeds "honestly". Intuitively, the reason that such a strategy succeeds with high probability is that in the real-world the parties cannot learn the output simultaneously; hence, one of our adversaries learns the output first with high probability. Since the adversary is able to detect the point in which it learns the output (using the extra key and tag), it can abort the computation, not allowing the honest party to learn the output.

We next formalize the above intuition by formally describing the two adversaries $\mathcal{A}$ and $\mathcal{B}$ mentioned above and analyzing their success probabilities. Assume by way of contradiction that $\mathbf{\Pi}$ is an r-round 3-party protocol that simultaneously guarantees full security against one fail-stop party and $\mathbf{1}/p$-security-with-abort against two fail-stop parties (for some $p(n) > \frac{1}{2} + \frac{1}{\mathbf{poly}}$).

The adversary $\mathcal{A}$: In each round i of the protocol, before broadcasting the messages of the corrupted p_1 and p_2, but after seeing the i'th message broadcast by p_3 (i.e., using the rushing property), the adversary $\mathcal{A}$ computes the value a_i as the output of honest parties p_1, p_2 where p_3 aborts after broadcasting the i'th message. If a_i is of the form (x, x') and $\mathbf{Ver}_{\mathbf{key}'_3}(x', \mathbf{tag}'_3) = \mathbf{1}$, then $\mathcal{A}$ instructs p_1 to abort and p_2 to proceed honestly with the computation, and outputs x'. Otherwise, both parties send their i'th message as prescribed by the protocol.

The adversary $\mathcal{B}$: The adversary $\mathcal{B}$ is defined analogously to $\mathcal{A}$. That is, in each round i of the protocol, before broadcasting the messages of the corrupted p_2 and p_3, but after seeing the i'th message broadcast by p_1, the adversary $\mathcal{B}$ computes the value b_i to be the output of honest parties p_2, p_3 in the case p_1 aborts after broadcasting the i'th message. If b_i is of the form (x, x') and $\mathbf{Ver}_{\mathbf{key}'_1}(x, \mathbf{tag}'_1) = \mathbf{1}$, then $\mathcal{B}$ instructs p_3 to abort and p_2 to proceed honestly with the computation, and outputs x. Otherwise, both parties send their i'th message as prescribed by the protocol.

By the full correctness of the protocol and by the way the inputs are chosen, it must hold that if no party aborts the execution, then the output of all parties is of the form (x, x') and that both $\mathbf{Ver}_{\mathbf{key}'_3}(x', \mathbf{tag}'_3) = \mathbf{1}$ and $\mathbf{Ver}_{\mathbf{key}'_1}(x, \mathbf{tag}'_1) = \mathbf{1}$. Hence, for every execution of the protocol, both $\mathcal{A}$ and $\mathcal{B}$ abort in some round (as the condition for abort holds in the last round). We can denote by $i_{\mathcal{A}}$ the round in which $\mathcal{A}$ aborts playing against an honest p_3, and by $i_{\mathcal{B}}$ the round in which $\mathcal{B}$ aborts playing against an honest p_1. Furthermore, either $i_{\mathcal{A}} \leq i_{\mathcal{B}}$ with probability at least $\mathbf{1/2}$ or $i_{\mathcal{A}} > i_{\mathcal{B}}$ with probability at least $\mathbf{1/2}$. Assume, without loss of generality, that the former holds. We argue that this means that $\mathcal{A}$ wins with probability negligibly close to $\mathbf{1/2}$. To show this is true we need to show that with probability close to $\mathbf{1/2}$ two things happen (1) $\mathcal{A}$ outputs x_3, and (2) p_3 does not output x_1. We consider all executions for which $i_{\mathcal{A}} \leq i_{\mathcal{B}}$ (occurring with probability close to 1/2), and we show that $\mathcal{A}$ wins in these executions, except with negligible probability. Since $\mathcal{A}$ aborts such executions in round $i_{\mathcal{A}} - \mathbf{1}$, that means that $\mathcal{A}$ outputs x' such that $\mathbf{Ver}_{\mathbf{key}'_3}(x', \mathbf{tag}'_3) = \mathbf{1}$. By the (information theoretic) security of the $\mathbf{MAC}$ scheme and since

the choice of $\mathbf{tag}'_3, \mathbf{key}'_3$ is independent of the execution of the protocol, except with negligible probability $x' = x_3$. On the other hand, since $i_{\mathcal{A}} \leq i_{\mathcal{B}}$, in this execution if p_1 aborts in round $i_{\mathcal{A}} - 1 \leq i_{\mathcal{B}} - 1$, then the output of p_2, p_3 is either $*$ or of the form (x, x') with $\mathbf{Ver}_{\mathbf{key}'_1}(x, \mathbf{tag}'_1) \neq 1$, namely, $x \neq x_1$.

We have shown a distribution over inputs on which one can distinguish the real-world from the ideal-world with probability $\frac{1}{2} - \mu(n)$, where μ is some negligible function. Hence, there must exist one set of inputs on which the same holds with at least this probability, contradicting the $1/p$-security against two fail-stop parties. □

7.2.2 Impossibility of Achieving "Best of Both Worlds" Computations with Non-Constant Number of Parties

Our best-of-both-worlds type protocols are more efficient when considering $1/p$-security-with-abort fall-back than when considering $1/p$-security fall-back. However, both cases inherit the limitation on the number of parties and the size of the domain and range from our $1/p$-secure protocols as discussed in Section 7. We next show that, in general, there is no efficient best-of-both-worlds type protocol (even with $1/p$-security-with-abort fall-back security) when the number of parties is $m(n) = \omega(1)$ and the size of the domain is polynomial, and when $m(n) = \omega(\log n)$ and the size of the domain of each party is 2. This is done using the 3-party impossibility proved in Lemma 7.1 (very similarly to the use of the impossibility result of Gordon and Katz [24] in Section 7.1).

We next state and prove our impossibility results.

Theorem 10. *For every $m(n) = \omega(\log n)$, there exists a deterministic $m(n)$-party functionality $\mathcal{F}'$ with domain $\{0,1\}$ that cannot be computed simultaneously guaranteeing full security with an honest majority and $1/p$-security-with-abort for $p \geq 2 + 1/\operatorname{poly}(n)$ without an honest majority.*

Furthermore, $\mathcal{F}'$ cannot even be computed when reducing the $1/p$-security-with-abort requirement to hold only against an adversary controlling $\left\lfloor \frac{m(n)}{2} \right\rfloor + 1$ parties.

Proof. Let $\mathcal{F} = \{f_n\}_{n \in \mathbb{N}}$ be the functionality guaranteed in Lemma 7.1 for $c \cdot \ell(n)$ for some constant c. Recall that the functionality $\mathcal{F}$ has two inputs of length $\ell(n)$, one for p_1 and one for p_3, and that p_2 has no input. Assume for simplicity that $m(n) = 2cm' + 1$. Define a $m(n)$-party (i.e., for parties $P_1, \ldots, P_{2cm'+1}$) deterministic functionality $\mathcal{F}' = \{f'_n\}_{n \in \mathbb{N}}$, where in f'_n party P_j for $1 \leq j \leq cm'$, gets the j-th bit of the input of p_1 in f_n, party P_j, for $cm' + 1 \leq j \leq 2cm'$, gets the $(j - cm')$-th bit of the input of p_3 in f_n, and $P_{2cm'+1}$ gets no input; the outputs of f_n and f'_n are equal. Assume towards a contradiction that $\mathcal{F}'$ can be computed simultaneously guarantee full security with an honest majority and $1/p$-security-with-abort without an honest majority by a protocol Π'. This implies a protocol Π for $\mathcal{F}$ with three parties that simultaneously guaranteeing full security with an honest majority and $1/p$-security-with-abort without an honest majority.

In protocol Π the first party (i.e., p_1) simulates the first cm' parties in Π', the third party (i.e., p_3) simulates the following cm' parties, and the second party (i.e., p_2) simulates the last party $P_{2cm'+1}$ in Π'.

- The full security of Π with an honest majority is implied by the fact that any adversary $\mathcal{A}$ for the protocol Π corrupting a single party p_k (where $k \in \{1,2,3\}$) can be transformed into an adversary $\mathcal{A}'$ for Π' controlling all parties that are simulated by p_k, which is a strict minority; as $\mathcal{A}'$ cannot violate the security of Π', the adversary $\mathcal{A}$ cannot violate the security of Π.

- The $1/p$-security-with-abort of Π against a dishonest majority is implied by the fact that any adversary $\mathcal{A}$ for the protocol Π corrupting two parties p_k, p_j (where $k, j \in \{1,2,3\}$) can be transformed

into an adversary $\mathcal{A}'$ for Π' controlling all parties that are simulated by p_k or by p_j; as $\mathcal{A}'$ cannot violate the $1/p$-security-with-abort of Π', the adversary $\mathcal{A}$ cannot violate the $1/p$-security of Π.

However, since such a 3-party protocol Π does not exist for $\mathcal{F}$, neither does Π'.

The further statement of the theorem is derived using the same arguments, looking further into the proof of Lemma 7.1. This proof shows that if a protocol Π for computing $\mathcal{F}$ is fully secure in the case of an honest majority, then its $1/p$-security-with-abort can either be attacked by an adversary corrupting p_1, p_2 or by an adversary corrupting p_2, p_3. Hence, any protocol Π' for computing $\mathcal{F}'$ that is fully secure in the case of an honest majority can either be attacked by an adversary corrupting $P_1, \ldots, P_{cm'}$ and $P_{2cm'+1}$ or by an adversary corrupting $P_{cm'+1}, \ldots, P_{2cm'}$ and $P_{2cm'+1}$. □

Theorem 11. *For every $m(n) = \omega(1)$, there exists a deterministic $m(n)$-party functionality $\mathcal{F}''$ with domain $\{0,1\}^{\log n}$ that cannot be computed simultaneously guaranteeing full security with an honest majority and $1/p$-security-with-abort for $p \geq 2 + 1/\operatorname{poly}(n)$ without an honest majority.*

Furthermore, $\mathcal{F}'$ cannot even be computed when reducing the $1/p$-security-with-abort requirement to hold only against an adversary controlling $\left\lfloor \frac{m(n)}{2} \right\rfloor + 1$ parties.

Proof. Assume for simplicity that $m(n) = 2cm' + 1$. Let $\ell(n) = cm' \log n$ and let $\mathcal{F} = \{f_n\}_{n \in \mathbb{N}}$ be the functionality guaranteed in Lemma 7.1 for $\ell(n)$. We divide the $\ell(n)$ bits of each input of f_n into cm' blocks of length $\log n$. Define an $(2cm' + 1)$-party deterministic functionality $\mathcal{F}'' = \{f''_n\}_{n \in \mathbb{N}}$, where in f''_n party p_j, for $1 \leq j \leq cm'$, gets the j-th block of $\log n$ bits of the input of p_1 in f_n, party p_j, for $cm' + 1 \leq j \leq 2cm' + 1$, gets the $(j - cm')$-th block of the input of p_3 in f_n, and the outputs of f_n and f'_n are equal. As in the proof of Theorem 10, a protocol Π'' that computes $\mathcal{F}''$ simultaneously guaranteeing full security with an honest majority and $1/p$-security-with-abort without an honest majority implies a protocol Π for $\mathcal{F}$ with three parties that simultaneously guaranteeing full security with an honest majority and $1/p$-security-with-abort without an honest majority contradicting Lemma 7.1. □

References

[1] G. Asharov, Y. Lindell, and T. Rabin. A full characterization of functions that imply fair coin tossing and ramifications to fairness. In *TCC*, pages 243–262, 2013.

[2] Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *J. of Cryptology*, 23(2):281–343, 2010.

[3] D. Beaver and S. Goldwasser. Multiparty computation with faulty majority. In *Proc. of the 30th IEEE Symp. on Foundations of Computer Science*, pages 468–473, 1989.

[4] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *Proc. of the 22nd ACM Symp. on the Theory of Computing*, pages 503–513, 1990.

[5] A. Beimel, E. Omri, and I. Orlov. Protocols for multiparty coin toss with dishonest majority. *J. of Cryptology*, 2013. To appear. Conference version in: T. Rabin, editor, Advances in Cryptology – CRYPTO 2010, volume 6223 of Lecture Notes in Computer Science, pages 538-557. Springer-Verlag, 2010.

[6] M. Ben-Or, O. Goldreich, S. Micali, and R. Rivest. A fair protocol for signing contracts. In *Proceedings of the 12th Colloquium on Automata, Languages and Programming*, pages 43–52. Springer-Verlag, 1985.

[7] M. Blum. How to exchange (secret) keys. *ACM Trans. Comput. Syst.*, 1(2):175–193, 1983.

[8] D. Boneh and M. Naor. Timed commitments. In M. Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 236–254. Springer-Verlag, 2000.

[9] R. Canetti. Security and composition of multiparty cryptographic protocols. *J. of Cryptology*, 13(1):143–202, 2000.

[10] R. Cleve. Limits on the security of coin flips when half the processors are faulty. In *Proc. of the 18th ACM Symp. on the Theory of Computing*, pages 364–369, 1986.

[11] R. Cleve. Controlled gradual disclosure schemes for random bits and their applications. In G. Brassard, editor, *Advances in Cryptology – CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 573–588. Springer-Verlag, 1990.

[12] I. Damgård. Practical and provably secure release of a secret and exchange of signatures. *J. of Cryptology*, 8(4):201–222, 1995.

[13] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *CACM*, 28(6):637–647, 1985.

[14] Z. Galil, S. Haber, and M. Yung. Cryptographic computation: Secure fault-tolerant protocols and the public-key model. In C. Pomerance, editor, *Advances in Cryptology – CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 135–155. Springer-Verlag, 1988.

[15] J. A. Garay, P. D. MacKenzie, M. Prabhakaran, and K. Yang. Resource fairness and composability of cryptographic protocols. *JCryptology*, 24(4):615–658, 2011.

[16] O. Goldreich. *Foundations of Cryptography, Voume I – Basic Tools*. Cambridge University Press, 2001.

[17] O. Goldreich. *Foundations of Cryptography, Voume II – Basic Applications*. Cambridge University Press, 2004.

[18] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proc. of the 19th ACM Symp. on the Theory of Computing*, pages 218–229, 1987.

[19] S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority. In A. J. Menezes and S. A. Vanstone, editors, *Advances in Cryptology – CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 77–93. Springer-Verlag, 1991.

[20] S. Goldwasser and Y. Lindell. Secure multi-party computation without agreement. *J. of Cryptology*, 18(3):247–287, 2005.

[21] S. D. Gordon, C. Hazay, J. Katz, and Y. Lindell. Complete fairness in secure two-party computation. *JACM*, 58(6):24, 2011.

[22] S. D. Gordon, Y. Ishai, T. Moran, R. Ostrovsky, and A. Sahai. On complete primitives for fairness. In D. Micciancio, editor, *Proc. of the Seventh Theory of Cryptography Conference – TCC 2010*, volume 5978 of *Lecture Notes in Computer Science*, pages 91–108. Springer-Verlag, 2010.

[23] S. D. Gordon and J. Katz. Complete fairness in multi-party computation without an honest majority. In O. Reingold, editor, *Proc. of the Sixth Theory of Cryptography Conference – TCC 2009*, Lecture Notes in Computer Science, pages 19–35, 2009.

[24] S. D. Gordon and J. Katz. Partial fairness in secure two-party computation. *J. of Cryptology*, 25(1):14–40, 2012.

[25] Y. Ishai, J. Katz, E. Kushilevitz, Y. Lindell, and E. Petrank. On achieving the "best of both world" in secure multiparty computation. *SIAM J. on Computing*, 40(1), 2011. Journal version of [26, 27].

[26] Y. Ishai, E. Kushilevitz, Y. Lindell, and E. Petrank. On combining privacy with guaranteed output delivery in secure multiparty computation. In *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 483–500. Springer-Verlag, 2006.

[27] J. Katz. On achieving the "best of both worlds" in secure multiparty computation. In *Proc. of the 39th ACM Symp. on the Theory of Computing*, pages 11–20, 2007.

[28] M. Luby, S. Micali, and C. Rackoff. How to simultaneously exchange a secret bit by flipping a symmetrically-biased coin. In *Proc. of the 24th IEEE Symp. on Foundations of Computer Science*, pages 11–21, 1983.

[29] T. Moran, M. Naor, and G. Segev. An optimally fair coin toss. In O. Reingold, editor, *Proc. of the Sixth Theory of Cryptography Conference – TCC 2009*, Lecture Notes in Computer Science, pages 1–18, 2009.

[30] R. Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In *Proc. of the 36th ACM Symp. on the Theory of Computing*, pages 232–241, 2004.

[31] B. Pinkas. Fair secure two-party computation. In E. Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 87–105. Springer-Verlag, 2003.

[32] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.

[33] A. C. Yao. How to generate and exchange secrets. In *Proc. of the 27th IEEE Symp. on Foundations of Computer Science*, pages 162–167, 1986.

A Proof of Lemma 2.6

Proof. Fix $\boldsymbol{D_1, D_2}$ satisfying Inequality (1). We prove the lemma by induction on $\boldsymbol{r}$. When $\boldsymbol{r = 1}$ the lemma is trivially true; Assume $\mathbf{win}(r+1) \leq 1/\alpha r + \beta$; we upper-bound $\mathbf{win}(r+1)$. As $\boldsymbol{\mathcal{A}}$ is unbounded, we can assume without loss of generality that $\boldsymbol{\mathcal{A}}$ is deterministic. Let $\boldsymbol{S}$ denote a set in the support of $\boldsymbol{D_2}$ such that $\boldsymbol{\mathcal{A}}$ aborts in the first iteration if and only if $\boldsymbol{a_1 \in S}$. We define $\boldsymbol{S_h}$ as all the elements $\boldsymbol{z \in S}$ s.t. $\mathbf{Pr}_{a \leftarrow D_1}[a = z] \geq \alpha \, \mathbf{Pr}_{a \leftarrow D_2}[a = z]$ holds for them and $\boldsymbol{S_\ell = S \setminus S_h}$. Observe that $\mathbf{Pr}[a_1 \in S_\ell] \leq \beta$. If $\boldsymbol{\mathcal{A}}$ does *not* abort in the first iteration, and the game does not end, then the

conditional distribution of $i^\star$ is uniform in $\{2, \ldots, r\}$ and the game $\Gamma(r+1)$ from this point forward is exactly equivalent to the game $\Gamma(r+1)$. In particular, conditioned on the game $\Gamma(r+1)$ not ending after the first iteration, the probability that $\mathcal{A}$ wins is at most $\mathbf{win}(r+1)$. We thus have

$$
\begin{aligned}
&\mathbf{Pr}[\mathbf{win}(r+1)] \\
&= \mathbf{Pr}[\mathcal{A}\,\mathbf{wins} \wedge a_1 \in S_\ell \wedge i^\star = 1] + \mathbf{Pr}[\mathcal{A}\,\mathbf{wins} \wedge a_1 \in S_h \wedge i^\star = 1] + \mathbf{Pr}[\mathcal{A}\,\mathbf{wins} \wedge i^\star > 1] \\
&\leq \mathbf{Pr}[a_1 \in S_\ell \wedge i^\star = 1] + \mathbf{Pr}[a_1 \in S_h \wedge i^\star = 1] + \mathbf{Pr}[\mathcal{A}\,\mathbf{wins} \wedge i^\star > 1] \\
&= \frac{\beta}{r+1} + \frac{1}{r+1} \Pr_{a_1 \leftarrow D_2}[a_1 \in S_h] + \frac{r}{r+1}\left(1 - \Pr_{a_1 \leftarrow D_1}[a_1 \in S]\right) \mathbf{win}(r) \\
&\leq \frac{\beta}{r+1} + \frac{1}{r+1} \Pr_{a_1 \leftarrow D_2}[a_1 \in S_h] + \frac{r}{r+1}\left(1 - \Pr_{a_1 \leftarrow D_1}[a_1 \in S_h]\right)\left(\frac{1}{\alpha r} + \beta\right) \\
&\leq \frac{\beta}{r+1} + \frac{1}{r+1} \Pr_{a_1 \leftarrow D_2}[a_1 \in S_h] + \frac{r}{r+1}\left(1 - \Pr_{a_1 \leftarrow D_1}[a_1 \in S_h]\right)\frac{1}{\alpha r} + \frac{r}{r+1}\beta \\
&\leq \beta + \frac{1}{r+1} \Pr_{a_1 \leftarrow D_2}[a_1 \in S_h] + \frac{r}{r+1}\left(1 - \alpha \Pr_{a_1 \leftarrow D_2}[a_1 \in S_h]\right)\frac{1}{\alpha r} \\
&= \beta + \frac{1}{\alpha(r+1)},
\end{aligned}
$$

□