Electronic Colloquium on Computational Complexity, Report No. 107 (2013)

Efficient Multiparty Protocols via Log-Depth Threshold Formulae

Gil Cohen* Ivan Bjerre Damgård† Yuval Ishai‡ Jonas Kölker†
Peter Bro Miltersen† Ran Raz* Ron D. Rothblum*

August 1, 2013

Abstract

We put forward a new approach for the design of efficient multiparty protocols:

1. Design a protocol π for a small number of parties (say, 3 or 4) which achieves security against a *single* corrupted party. Such protocols are typically easy to construct, as they may employ techniques that do not scale well with the number of corrupted parties.
2. Recursively compose π with itself to obtain an efficient n-party protocol which achieves security against a constant fraction of corrupted parties.

The second step of our approach combines the "player emulation" technique of Hirt and Maurer (J. Cryptology, 2000) with constructions of logarithmic-depth formulae which compute threshold functions using only constant fan-in threshold gates.

Using this approach, we simplify and improve on previous results in cryptography and distributed computing. In particular:

- We provide conceptually simple constructions of efficient protocols for Secure Multiparty Computation (MPC) in the presence of an honest majority, as well as broadcast protocols from point-to-point channels and a 2-cast primitive.
- We obtain new results on MPC over blackbox groups and other algebraic structures.

The above results rely on the following complexity-theoretic contributions, which may be of independent interest:

*Weizmann Institute. {gil.cohen,ran.raz,ron.rothblum}@weizmann.ac.il. Supported by ISF (Israel Science Foundation) grants and by the I-CORE Program of the Planning and Budgeting Committee.

†Aarhus University. {ivan,epona,bromille}@cs.au.dk. Supported by the Danish National Research Foundation and The National Science Foundation of China (under the grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation, within which part of this work was performed; and also from the CFEM research center (supported by the Danish Strategic Research Council) within which part of this work was performed.

‡Technion. yuvali@cs.technion.ac.il. Supported by the European Research Council as part of the ERC project CaC (grant 259426).

ISSN 1433-8092

- We show that for every $j, k \in \mathbb{N}$ such that $m \triangleq \frac{k-1}{j-1}$ is an integer, there is an explicit (poly(n)-time) construction of a logarithmic-depth formula which computes a good approximation of an (n/m)-out-of-n threshold function using only j-out-of-k threshold gates and no constants.
- For the special case of n-bit majority from 3-bit majority gates, a non-explicit construction follows from the work of Valiant (J. Algorithms, 1984). For this special case, we provide an explicit construction with a better approximation than for the general threshold case, and also an *exact* explicit construction based on standard complexity-theoretic or cryptographic assumptions.

Contents

1 Introduction

Secure multiparty computation (MPC) enables a set of parties to jointly accomplish some distributed computational task, while maintaining the secrecy of the inputs and the correctness of the outputs in the presence of coalitions of dishonest parties. Originating from the seminal works of [Yao82a, GMW87, BGW88, CCD88], secure MPC has been the subject of an enormous body of work.

Despite this body of work, MPC protocols remain quite complicated and their security is difficult to prove. In this work we propose a new general approach to the construction of efficient[1] multiparty protocols in the presence of an honest majority. This approach enables us to obtain conceptually simple derivations of known feasibility results (or slightly weaker variants of such results), and also to obtain new results.

Our approach is inspired by, and builds on, the "player emulation" technique of Hirt and Maurer [HM00], who obtain secure MPC protocols by reducing the construction of an n-party protocol to the task of constructing a protocol π for a constant (e.g., three or four) number of parties. The motivation of [HM00] was to obtain n-party protocols that are secure with respect to general (non-threshold) adversary structures. A disadvantage of their n-party protocols is that their complexity grows exponentially with n. This seems inevitable when considering arbitrary adversary structures.

Our motivation is very different: We would like to use the atomic protocol π for constructing *efficient* n-party protocols in the traditional MPC setting of *threshold* adversary structures. Since π only involves a small number of parties, its design may employ simpler techniques that do not scale well with the number of corrupted parties. Thus, our goal is to simplify the design of efficient n-party protocols by reducing it to the design of a simpler atomic protocol π.

To make the approach of [HM00] scale with the number of parties, we introduce a new complexity-theoretic primitive: a logarithmic-depth formula[2] which is composed only of constant-size threshold gates and computes an n-input threshold function. The problem of constructing such formulae is closely related to a classical problem in complexity theory. In this work we also make a contribution to this complexity-theoretic problem, which may be of independent interest.

In addition to providing conceptually simple protocols, our approach is very general and can be applied in a variety of settings and models. In contrast to most traditional MPC protocols, it is not tied to some underlying algebraic structure. We demonstrate this generality by obtaining new results on MPC over black-box groups and other algebraic structures, improving on previous results from the literature.

Before proceeding to describe the details of our approach, we note that the goal of designing MPC protocols whose complexity grows (only) polynomially with the number of parties also has relevance to *two-party* cryptography. Indeed, there are general techniques for applying MPC protocols with security in the presence of an honest majority (where the number of parties grows with the security parameter) towards two-party tasks such

[1]Here and throughout this work, by "efficient" we mean polynomial-time in the number of parties and the input size.

[2]A formula is a circuit with fan-out 1. A logarithmic-depth formula (more precisely, infinite family of formulas) is one whose depth is $O(\log n)$, where n is the number of inputs. Throughout this paper we consider only *monotone* formulas without negations or constants.

as zero-knowledge proofs and secure two-party computation [IKOS09, IPS08].

1.1 Our Approach

In the following, for simplicity, we consider the case of perfect security against a *passive* adversary. In this setting, parties are honest but curious. That is, they follow the protocol but may attempt to learn secret information based on what they see. We note that, in contrast to the norm, the extension of this approach to the case of an active adversary is relatively straightforward.

We first give an overview of the player emulation technique of Hirt and Maurer [HM00] and then proceed to describe how we overcome the exponential blow-up incurred by [HM00] in the case of threshold adversary structures.

Recall that security of MPC protocols is defined by comparing a real protocol to an ideal protocol, in which, in addition to the parties involved in the computation, there is a trusted party. A protocol is deemed secure if for every adversary in the real protocol controlling a subset of the parties, there is an equivalent adversary controlling the same subset in the ideal protocol.

The technique from [HM00] is to reduce the design of n-party protocols to the design of protocols that support only 3 parties (the minimal number of parties for perfect security in the passive security model).

We proceed to present an informal description of the reduction. Indeed, suppose that the 3-party case has been solved. That is, for every computational task involving three parties there exists a secure protocol that securely implements this task when at most one of the parties is passively corrupted.[3] We describe how to use this protocol to securely implement computational tasks using a larger number of parties.

Consider n parties that wish to securely accomplish some joint computational task. It is best to think of this task as being specified by an ideal protocol π_0 which involves, in addition to the n parties, a trusted party τ. The ideal protocol is secure (by definition) even if the adversary controls any subset of the parties that does not contain τ.

Consider a new protocol π_1 that involves the n original parties but where we replace the trusted party τ with three new virtual parties v_1, v_2, v_3. Since in π_0, the trusted party τ is just involved in a computational task, we can use the given 3-party protocol to simulate τ using v_1, v_2, v_3. When is the new protocol π_1 secure? Since π_0 was only insecure whenever the adversary controlled τ and since the 3-party protocol is secure as long as the adversary controls at most one of the virtual parties, π_1 is secure as long as the adversary does not control two or more of the virtual parties.

We continue this process by designing a new protocol π_2 in which the virtual party v_1 is itself simulated by three new virtual parties w_1, w_2, w_3. Since π_1 is only insecure whenever the adversary controls more than one of v_1, v_2, v_3 and since the protocol for emulating v_1 is secure when at most one of w_1, w_2, w_3 is controlled by the adversary, π_2 is secure as long as the adversary does not control either v_2 and v_3 or one of v_2, v_3 and two or more of w_1, w_2, w_3.

We continue in this process simulating virtual parties by more virtual parties. The sets of corrupted parties against which the resulting protocol is secure can be described

[3]Since we deal with *perfect* security, the size of the secure protocol depends only on the size of the original protocol. In particular, any constant size protocol can be implemented securely in constant size.

by looking at a formula composed of 3-input majority gates which we denote by Maj_3. Each wire represents a virtual party. The protocol π_1 can be represented by a simple formula F_1 consisting of a single Maj_3 gate where the three input wires correspond to the virtual parties v_1, v_2, v_3 and the output wire corresponds to τ. We assign to each input wire corresponding to an honest party a value of 0 and a value of 1 to those corresponding to dishonest parties. It can be easily verified that the protocol is secure whenever the formula F_1 evaluates to 0.

Similarly, the protocol π_2 can be represented by a formula F_2 which is constructed from F_1 by connecting the input wire corresponding to v_1 with an additional Maj_3 gate with three new input wires (corresponding to w_1, w_2, w_3). It is easy to verify that the new protocol is secure whenever the formula evaluates to 0.

Suppose that we continue on like this but instead of arbitrarily choosing which virtual party to simulate, we choose it according to some formula F, composed only of Maj_3 gates.[4] Once we reach the input layer of the formula, we associate each input variable to a real party and every remaining virtual party is simulated by the real party associated with the corresponding input wire.

As above, the protocol is secure against every set T of parties on which the formula F evaluates to 0. (Here and in the following we associate a set T with its characteristic vector χ_T.) Thus, to obtain a protocol that is secure for a particular adversary structure, it suffices to provide a formula that evaluates to 0 on all sets in the structure. Since, in contrast to [HM00], our goal is merely to obtain security in the presence of an honest majority, we need only to construct a formula that computes the majority function (using only Maj_3 gates and no constants).

Such a formula was implicitly constructed by Hirt and Maurer [HM00] for general Q_2 functions[5] and in particular for majority. Unfortunately, the formula of [HM00] has linear depth. This yields a protocol whose complexity grows exponentially with the number of parties, since when traversing the formula we increased the complexity of the protocol by a constant multiplicative factor (corresponding to the number of operations in the 3-party protocol) at every layer.

To overcome the exponential blowup, we replace the formula of [HM00] by a *logarithmic-depth* formula (which computes the majority function using only Maj_3 gates). Using the formula-based protocol described above, the logarithmic depth results in an efficient protocol, namely one whose complexity only grows polynomially with the number of parties. In Section 1.2 we describe the construction of a good "approximation" of such a formula as well as exact constructions under standard complexity-theoretic assumptions.

This approach is indeed very general and can be used in different models of secure MPC. For example, it can be used to obtain both passive security as outlined above and active security by using an underlying 4-party protocol that is secure against one active party and a log-depth threshold formula composed of two-out-of-four threshold gates (denoted by Th_2^4) which we also construct (see Section 1.2).

In fact, this reduction gives us a "cookbook" for designing secure multiparty protocols. The first step is to design a protocol for a constant number of parties that is secure

[4]Actually, [HM00] do not present their construction in the terminology of Maj_3 formulae; we use this presentation since it is more intuitive and is better suited for our purposes.

[5]A monotone function $f : \{0,1\}^n \to \{0,1\}$ is said to be of type Q_d if $f(x_1) = f(x_2) = \ldots = f(x_d) = 0$ implies that $x_1 \vee x_2 \vee \ldots \vee x_d \neq 1^n$.

against one dishonest party and the second step is to use a logarithmic-depth threshold from thresholds formula to obtain an efficient multiparty protocol that is secure against a constant fraction of corrupted parties.

We demonstrate the generality of this approach by deriving protocols in both passive and active settings and in different MPC models which differ in the type of underlying algebraic structure, including models for which no protocols were known. We also obtain conceptually simple protocols for classical problems in distributed computing such as broadcast protocols.

Simplified feasibility results. The classical results of Ben-Or et al. [BGW88] and Chaum et al. [CCD88] allow n parties to evaluate an arbitrary function, using secure point-to-point channels, with perfect security against $t < n/2$ passively corrupted parties or $t < n/3$ actively corrupted parties. We can derive conceptually simpler variants of these results by applying our approach with π being a 3-party or 4-party instance of the simple MPC protocol of Maurer [Mau06]. On the one hand our results are slightly weaker because they either need the threshold t to be slightly sub-optimal or alternatively require (standard) complexity theoretic assumptions to construct an appropriate formula for implementing the protocol. It is instructive to note that the complexity of Maurer's protocol grows exponentially with the number of parties. Our approach makes this a non-issue, as we only use the protocol from [Mau06] with a constant number of parties.[6]

MPC over blackbox algebraic structures. There has been a considerable amount of work on implementing MPC protocols for computations over different algebraic structures such as fields, rings, and groups. Algebraic computations arise in many application scenarios. While it is possible in principle to emulate each algebraic operation by a sequence of boolean operations, this is inefficient both in theory and in practice. In particular, the communication complexity of the resulting protocols grows with the computational complexity of the algebraic operations rather than just with the bit-length of the inputs and outputs. This overhead can be avoided by designing protocols which make a *black-box* (i.e., oracle) use of the underlying structure. The advantage of such protocols is that their communication complexity and the number of algebraic operations they employ are independent of the complexity of the structure.

MPC over rings and k-linear maps. The work of Cramer *et al.* [CFIK03] shows how to efficiently implement secure MPC over blackbox *rings*. We obtain a simpler derivation of such a protocol by noting that the simple protocol of Maurer [Mau06] directly generalizes to work over a blackbox ring. As before, one could not apply this protocol directly because its complexity is exponential in the number of parties. We show how to use a similar approach for obtaining the first blackbox feasibility results for MPC over *k-linear maps*. See Section 10 for details.

[6]While in the present work we apply our approach only to perfectly secure protocols, one could apply a similar technique to derive the result of Rabin and Ben-Or [RBO89], namely a statistically secure protocol which tolerates $t < n/2$ actively corrupted parties.

MPC over groups. The problem of MPC over blackbox *groups* was introduced by Desmedt *et al.* [DPSW07] and further studied in [SYT08, DPS$^+$12b, DPS12a]. [7] To apply our approach in the group model, we need to specify the atomic protocol π that we use. For the case of passive security, we directly construct a simple 3-party protocol that has security against one corrupted party. This protocol is loosely based on a protocol by Feige *et al.* [FKN94] and considerably simplifies the 3-party instance of a general result from [DPS$^+$12b].

In the active security model, we rely on the recent work of [DPS12a] who obtain the first MPC protocols with active security in the group model. The complexity of the protocol of [DPS12a] grows exponentially with the number of parties. However, we only need to employ the [DPS12a] protocol for four parties and so we do not suffer the exponential blowup. Thus, we settle the main problem left open in [DPS12a] by applying our technique to an instance of their results.

We also obtain the first *two-party* MPC protocols over blackbox groups. In the passive corruption model, we combine a group product randomization technique due to Kilian [Kil88] with a "subset sum" based statistical secret sharing of group elements. We then get security against active corruptions by combining this two-party protocol with our efficient n-party protocol for the active model via the IPS compiler [IPS08]. See Section 11 for details.

Broadcast. Broadcast is one of the most basic problems in distributed computing. Recall that in a broadcast protocol a broadcaster wants to send a message to all other parties. A broadcast protocol should end with all parties holding the same value, even if some of the parties, possibly including the broadcaster, behave adversarially. Obtaining efficient broadcast protocols is a highly nontrivial task [PSL80, Dol82, GM98]. Our generic approach for MPC protocols can be used to directly construct simple broadcast protocols for $t < n/3$ corrupted parties. We also get a simplified proof of a result of Fitzi and Maurer [FM00], showing that an ideal primitive allowing broadcast for 3 parties (so-called 2-cast) implies broadcast with $t < n/2$ corrupted parties. Our proof technique also yields broadcast for the more general case of Q_2 adversaries which was previously an open problem. See Section 7 for details.

1.2 Threshold Formulae from Threshold Gates

Motivated by the above applications to MPC, we consider the problem of constructing a logarithmic-depth threshold formula from threshold gates. Before discussing the general problem, we first discuss the special case of constructing a logarithmic depth formula composed of Maj_3 gates that computes the majority function. Note that this is exactly the type of formula required in the setting of *passive* MPC security.

[7] Interestingly, group-based MPC with low security threshold was implicitly used in the recent work of Miles and Viola on leakage-resilient circuits [MV13]. It seems likely that efficient group-based MPC protocols with near-optimal security threshold, such as those obtained in our work, can be useful in this context.

1.2.1 Majority from Majorities.

A closely related problem was considered by Valiant [Val84] who proved the existence of a logarithmic-depth monotone formula that computes the majority function where the formula uses And and Or gates, both of fan-in 2. As noted independently by several authors [BM92, GM96, Zwi96, Gol11b], a slight modification of Valiant's argument shows the existence of a logarithmic-depth formula composed of Maj_3 gates that computes the majority function.

Valiant's proof is based on the probabilistic method and is non-constructive. Namely, the proof only assures us of the existence of a formula with the above properties, but does not hint on how to find it efficiently. Motivated by the applications presented in Section 1.1, we ask whether Valiant's proof can be derandomized using only Maj_3 gates and no constants.[8] We raise the following conjecture:

Conjecture 1 (Majority from Majorities)**.** *There exists an algorithm A that given an odd integer n as input, runs in $\mathrm{poly}(n)$-time and generates a formula F on n inputs, with the following properties:*

- *F consists only of Maj_3 gates and no constants.*
- $\mathsf{depth}(F) = O(\log n)$.
- *F computes the majority function on n inputs.*

A derandomization for Valiant's proof for formulas over And and Or gates follows from the seminal paper of Ajtai, Komlós and Szemerédi [AKS83], though the latter does not seem to imply a derandomization in the context of Maj_3 gates, where constants are not allowed.[9]

In this paper we make a significant progress towards proving Conjecture 1. In particular, we prove that relaxed variants of the conjecture hold. In addition, we show that the conjecture follows from standard complexity assumptions, namely, $\mathsf{E} \triangleq \mathsf{DTIME}(2^{O(n)})$ does not have $2^{\varepsilon n}$-size circuits for some constant $\varepsilon > 0$. Note that the latter follows from the existence of exponentially hard one-way functions.[10] See details in Section 2.

1.2.2 Threshold Formulae from Threshold Gates.

Motivated by applications to the active MPC setting, and being a natural complexity-theoretic problem on its own, we initiate the study of a generalization of the majority from majorities problem, which we call *the threshold from thresholds problem.*

For integers $2 \le j \le k$, define the threshold function $\mathsf{Th}^k_j : \{0,1\}^k \to \{0,1\}$ as follows. $\mathsf{Th}^k_j(x) = 1$ if and only if the Hamming weight of x is at least j. Note that $\mathsf{Maj}_3 = \mathsf{Th}^3_2$.

Unlike the majority from majorities problem, it is not a priori clear what threshold function, if any, can be computed by a log-depth formula composed only of Th^k_j gates, even if no explicit construction is required.

[8] We cannot allow the use of the constant 0, as this would correspond to assuming parties to be incorruptible. The use of the constant 1 alone is not helpful in our context.

[9] Note that And and Or gates can be implemented using Maj_3 gates and constants.

[10] We find it curious that *perfectly secure* MPC results are based on the existence of (sufficiently strong) one-way functions.

We make significant progress also on this question. Roughly speaking, we provide an explicit construction of a logarithmic depth formula composed solely of Th_j^k gates, that well approximates $\mathsf{Th}_{n/m}^n$, where $m = \frac{k-1}{j-1}$. For further details, see Section 2.3.

1.3 Organization

In Section 2 we state our results and in Section 3 we present the proof techniques of the complexity-theoretic part (for an overview of applications to cryptography and distributed computing, see Section 1.1).

Being of potential interest to researchers in cryptography, complexity theory and distributed computing, the technical part of the paper is partitioned into three corresponding and stand-alone parts.

The complexity theoretic part encompasses Sections 4 to 6. In Section 4 we give basic definitions relevant only to this part of the paper. In Section 5 we present our constructions of threshold formulae from threshold gates. In Section 6 we address the special case of majority formulae from majority gates.

The distributed computing part is presented in Section 7.

The cryptographic part encompasses Sections 8 to 11. In Section 8 we define multi-party computation. In Section 9 we present our generic approach for constructing MPC protocols from threshold formulae. In Section 10 we show how to obtain results in the ring MPC model. In Section 11 we show new results in the group MPC model.

2 Our Results

We first describe the applications of our approach in cryptography and distributed computing, and then proceed to the complexity-theoretic results.

2.1 Cryptographic Results

We start by stating known results that we rederive using our approach, and later state our new results.

In the passive Ring-MPC model, we get the following results.

- If the majority from majorities conjecture (Conjecture 1) holds then we obtain an explicit MPC protocol that has optimal security in the passive model. That is, it is secure as long as at most a $\frac{1}{2} - \Omega(\frac{1}{n})$ fraction of the n parties (more precisely, $t < n/2$) are passively corrupted. See Theorem 13.

  As noted above and stated formally in Theorem 3, Conjecture 1 follows from widely-believed conjectures in complexity theory and cryptography.

- An unconditional explicit and close to optimal protocol in the passive model in which the fraction of dishonest parties is at most $\frac{1}{2} - 2^{-O(\sqrt{\log n})}$ out of the n parties (in contrast to the optimal threshold of $\frac{1}{2} - \Omega(\frac{1}{n})$). See Theorem 14.

- A randomized construction of an optimal protocol in the passive model. By randomized construction we mean that the protocol is constructed by a randomized algorithm which may fail with negligible (undetectable) probability, but otherwise outputs the description of a perfect protocol. See Theorem 15.

We obtain the following result in the *active* Ring-MPC model.

- An explicit but non-optimal protocol that is secure against any *active* adversary that controls at most a $\frac{1}{3} - \Omega(\frac{1}{\sqrt{\log n}})$ fraction of the n parties (in contrast to the optimal bound of $\frac{1}{3} - \Omega(\frac{1}{n})$). See Theorem 17.

Next we state our new results in the blackbox group model, introduced by Desmedt *et al.* [DPSW07, DPS+12b]. In this model the function computed by the protocol is specified by an arithmetic circuit over a (possibly non-Abelian) group, and the parties are restricted to making blackbox access to the group. (This includes oracle access to the group operation, taking inverses, and sampling random group elements; see Section 11.) In particular, the number of group operations performed by the protocol should not depend on the structure of the group or the complexity of implementing a group operation using, say, a Boolean circuit.

- **Group-MPC, passive:** The best explicit protocol of [DPS+12b] offers perfect security against a $\frac{1}{n^\epsilon}$ fraction of passively corrupted parties, for any constant $\epsilon > 0$, where n is the total number of parties.

  We improve upon the latter by constructing an explicit protocol that has perfect security against an (almost optimal) $\frac{1}{2} - 2^{-O(\sqrt{\log n})}$ fraction of passively corrupted parties (see Theorem 19). Alternatively, we get an optimal bound of $\frac{1}{2} - \Omega(\frac{1}{n})$ assuming the majority from majorities conjecture (see Theorem 20), via a non-uniform construction (see Theorem 21), or under standard derandomization or cryptographic assumptions. Lastly, we also obtain a protocol with an optimal bound of $\frac{1}{2} - \Omega(\frac{1}{n})$ with a running time that is only quasi-polynomial in the number of parties (see Theorem 22).

- **Group-MPC, active:** In a recent work, Desmedt *et al.* [DPS12a] constructed a secure MPC protocol in the group model with security against an *active* adversary. However, their result only gives a protocol whose complexity depends exponentially on the number of parties, regardless of the corruption threshold.

  We construct an *efficient* secure MPC protocol in the group model where an active adversary can control (an almost optimal) $\frac{1}{3} - \Omega(\frac{1}{\sqrt{\log n}})$ fraction of the n parties.

- **Secure two-party computation over groups:** We construct the first secure *two-party* protocols over blackbox groups. Our protocols offer statistical security against active corruptions (assuming an oblivious transfer oracle) and rely on the afforementioned n-party protocols over black-box groups.

Finally, our protocols for the Ring-MPC model described above can be generalized to yield the following new result for MPC over k-linear maps.

- **MPC over k-linear maps:** We show that, for any constant k and any basis B of k-linear maps over finite Abelian groups, there are efficient MPC protocols for computing circuits over B which only make blackbox access to functions in B and group operations. This generalizes previous results for MPC over blackbox rings [CFIK03], which follow from the case $k = 2$, and can potentially be useful in cryptographic applications that involve complex bilinear or k-linear maps. These protocols are perfectly secure against a $\frac{1}{k} - \Omega(\frac{1}{\sqrt{\log n}})$ fraction of passively corrupted parties or a $\frac{1}{k+1} - \Omega(\frac{1}{\sqrt{\log n}})$ fraction of actively corrupted parties.

2.2 Distributed Computing Results

Broadcast. It is well known that broadcast can be implemented over point-to-point channels if and only if less than a third of the parties are actively corrupted [PSL80, Dol82] or, more generally, if and only if no three of the subsets the adversary may corrupt cover the entire set of parties [HM00, FM98], a so called Q_3-adversary.

In this paper we show that a trivial broadcast protocol for 4 parties where one is actively corrupted easily implies the result of [FM98] using existing constructions of (super-logarithmic depth) formulae. Substituting instead our own logarithmic depth formula constructions implies a simple polynomial-time broadcast protocol for less than $n(\frac{1}{3} - \Omega(\frac{1}{\sqrt{\log n}}))$ corrupted parties.

Broadcast from 2-cast. In [FM00], Fitzi and Maurer identify a minimal primitive that allows to improve the $\frac{n}{3}$ corruption threshold: if we are given the ability to broadcast among any subset of 3 parties for free, a so-called *2-cast* primitive, then broadcast becomes possible when less than $\frac{n}{2}$ parties are corrupted. It is natural to ask whether 2-cast also implies broadcast secure against general Q_2-adversaries (where no two corruptible subsets cover the entire set of parties). This problem was previously open.

We apply our approach to construct broadcast protocols based on a 2-cast primitive. Together with existing constructions of (super-logarithmic depth) formulae composed of Maj_3-gates, this immediately implies a construction of broadcast from 2-cast for every Q_2-adversary, resolving the above problem. Substituting instead our logarithmic-depth formula constructions, we get a simplified derivation of polynomial-time protocols for the case of an honest majority considered in [FM00]. We do not know if the formula based approach also implies the results in [CFF$^+$05], which consider generalizations of the 2-cast primitive.

2.3 Complexity-Theoretic Results

In this section we describe our results on constructing threshold formulae from threshold gates. For the special case of computing majority from Maj_3 gates we obtain stronger results which we state first.

2.3.1 Majority from Majorities

Our first complexity-theoretic result shows that given a small promise on the bias of the input (defined as the difference between the normalized Hamming weight and 1/2), Conjecture 1 holds.

Theorem 2. *There exists an algorithm A that given an odd integer n as input, runs in* $\mathrm{poly}(n)$*-time and computes a formula F on n inputs, with the following properties:*

- *F consists only of Maj_3 gates and no constants.*
- $\mathsf{depth}(F) = O(\log n)$.
- *$\forall x \in \{0,1\}^n$ such that $\mathsf{bias}(x) \geq 2^{-O(\sqrt{\log n})}$ it holds that $F(x) = \mathsf{Maj}(x)$.*

We note that the proof of Theorem 2 also gives a construction of a formula that computes the majority function *exactly* (i.e., without a promise on the bias) but with depth that is only poly-logarithmic (rather than logarithmic), see Lemma 6.1.

Our second result shows that under standard complexity hardness assumptions, Conjecture 1 holds.

Theorem 3. *If there exists an $\varepsilon > 0$ such that $\mathsf{E} \triangleq \mathsf{DTIME}(2^{O(n)})$ does not have $2^{\varepsilon n}$-size circuits then Conjecture 1 holds. In particular, if there exist exponentially hard one-way functions then Conjecture 1 holds.*[11]

In fact, the proof of Theorem 3 explicitly presents an algorithm for constructing a formula as in Conjecture 1 given the truth table of any function in E, on a suitable number of inputs, that cannot be computed by $2^{\varepsilon n}$-size circuits. Moreover, the assumption made in Theoerem 3 can be relaxed (see the end of Section 6.2).

2.3.2 Thresholds Formulae from Threshold Gates

Lemma 4. *There exists an algorithm A that given $t, j, k \in \mathbb{N}$ as input, where j, k are constants in t such that $j \geq 2$ and $k \geq 2j-1$,*[12] *runs in $\exp(t)$-time and generates a formula F with the following properties:*

- *F has $mt+1$ inputs, where $m = \left\lfloor \frac{k-1}{j-1} \right\rfloor$.*
- *F consists only of Th^k_j gates and no constants.*
- $\mathsf{depth}(F) = O(t)$.
- *$\forall x \in \{0,1\}^{mt+1}$ it holds that $F(x) = \mathsf{Th}^{mt+1}_{t+1}(x)$.*

Lemma 4 generalizes results of [AR63, HM00, BIW10], who proved it for particular values of j and k, and uses a similar technique. We note that the depth of the formula generated in Lemma 4 is linear, which is too large for our applications. Nevertheless, the following theorem, which uses Lemma 4 as a building block, shows that a formula with *logarithmic depth* can be generated efficiently assuming a sufficient "bias" on the input.

[11] A one-way function f is *exponentially hard* if there exists an $\varepsilon > 0$ such that every family of $2^{\varepsilon n}$-size circuits can invert f with only $2^{-\varepsilon n}$ probability. If there exists such a function f, then the language $\mathcal{L}_f$ is in E but does not have $2^{\varepsilon n}$-size circuits, where $\mathcal{L}_f = \{(y, x', 1^n) : y$ has a preimage of length n under f which starts with $x'\}$.

[12] Throughout the paper we assume, without loss of generality, that $k \geq 2j-1$. The complementary case can be reduced to this one by using Th^k_{k-j+1} gates and interpreting 0 as 1 and vice versa.

Theorem 5. *There exists an algorithm A that given $n, j, k \in \mathbb{N}$ as input, where j, k are constants in n such that $j \geq 2$ and $k \geq 2j - 1$, runs in $\mathrm{poly}(n)$-time and generates a formula F on n inputs, with the following properties:*

- *F consists only of Th_j^k gates and no constants.*
- $\mathsf{depth}(F) = O(\log n)$.
- *$\forall x \in \{0,1\}^n$ with normalized Hamming weight at least $\frac{1}{m} + \Omega(\frac{1}{\sqrt{\log n}})$, it holds that $F(x) = 1$, where $m = \left\lfloor \frac{k-1}{j-1} \right\rfloor$.*
- *$\forall x \in \{0,1\}^n$ with normalized Hamming weight at most $\frac{1}{m} - \Omega(\frac{1}{\sqrt{\log n}})$, it holds that $F(x) = 0$.*

Note that Theorem 2 is not a special case of Theorem 5 (with $j = 2, k = 3$) as the required promise on the bias in Theorem 2 is exponentially smaller than that in Theorem 5.

We do not know whether an analog of Conjecture 1 is plausible for the threshold from thresholds problem, even without the time-efficiency requirement. Theorem 5 might serve as evidence for the affirmative. However, the probabilistic argument used in the majority from majorities problem (see, e.g., [Gol11b]) breaks for this more general case. We consider this to be an interesting open problem for future research.

3 Proof Overview of Complexity-Theoretic Results

In this section we give an overview of our complexity-theoretic constructions. For simplicity, we start by giving an overview of our construction of a logarithmic-depth formula composed of Maj_3 gates, and no constants, that computes the majority function for inputs with constant bias. That is, we informally describe an efficient algorithm that given n, ε as inputs, where $\varepsilon > 0$ is constant in n, outputs a logarithmic-depth formula with n inputs which computes the majority function correctly on inputs with bias at least ε. It is not hard to see (see Observation 4.1) that it is enough to construct a logarithmic-depth *circuit*, since such a circuit can be efficiently converted to an equivalent logarithmic-depth formula.

To this end, we design an algorithm called $\mathsf{ShrinkerGenerator}$ (see Lemma 5.5) that given n, ε as inputs, generates a *constant-depth* circuit $\mathsf{Shrinker}$ with n inputs and $\frac{n}{2}$ outputs, composed of Maj_3 gates and no constants, such that

$$\forall x \in \{0,1\}^n \quad \mathsf{bias}(x) \geq \varepsilon \implies \mathsf{bias}(\mathsf{Shrinker}(x)) \geq \varepsilon.$$

Thus, $\mathsf{Shrinker}$ shrinks the number of variables to half while maintaining the bias, assuming the input has a sufficiently large bias. By repeatedly calling $\mathsf{ShrinkerGenerator}$ on inputs $n, \frac{n}{2}, \frac{n}{4}, \ldots, 2$ (with the same ε) and concatenating the resulting circuits, one gets a logarithmic-depth circuit that computes the majority function assuming the input has large enough bias.

A key object we use in the design of $\mathsf{ShrinkerGenerator}$ is a Boolean sampler. Roughly speaking, a Boolean sampler is a randomized algorithm which on input $x \in \{0,1\}^n$

approximates the Hamming weight of x by reading only a small number of the bits of x. More precisely, a (d, ε, δ)-Boolean sampler is a randomized algorithm that on input $x \in \{0,1\}^n$ with normalized Hamming weight ω, samples at most d bits of x, and outputs $\beta \in [0,1]$ such that $\mathsf{Pr}[|\omega - \beta| \geq \varepsilon] \leq \delta$.

We will use a special type of samplers which take their samples in a non-adaptive fashion, and their output is simply the average of the sampled bits. For any $\varepsilon, \delta > 0$ there exist efficient (d, ε, δ)-Boolean samplers, with $d = O(\varepsilon^{-2} \cdot \delta^{-1})$, that on inputs of length n use only $\log n$ random bits (see Theorem 6).

Because such a sampler is non-adaptive and simply outputs the average of the sampled bits, it can be represented as a bipartite graph $G = (L, R, E)$, with $|L| = |R| = n$. For an input $x \in \{0,1\}^n$, the i'th vertex in L is labeled with the i'th bit of x. Each vertex in R represents one of the possible $\log n$ bit random strings used by the sampler. Each right vertex r is connected to the d left-vertices that are sampled by the algorithm when r is used as the random string.

The algorithm $\mathsf{ShrinkerGenerator}$ on inputs n, ε starts by constructing a graph G that represents a $(d, \frac{\varepsilon}{2}, \frac{1}{8})$-Boolean sampler, with $d = \mathrm{poly}(\frac{1}{\varepsilon}) = O(1)$. It then arbitrarily chooses half of the right vertices in G and discards the rest. This gives a bipartite graph $G' = (L', R', E')$ with $|L'| = n$, $|R'| = \frac{n}{2}$ and constant right-degree d. The circuit $\mathsf{Shrinker}$ that the algorithm $\mathsf{ShrinkerGenerator}$ outputs is given by placing a circuit that computes the majority function on d inputs for every right vertex. The inputs of this majority circuit are the neighbors of the respective right vertex. Note that as d is constant, a constant-depth circuit that computes the majority function on d inputs can be found in constant time.

As for the correctness of the construction, assume now that $x \in \{0,1\}^n$ has some constant bias ε and, without loss of generality, assume that the bias is towards 1 (i.e., $\mathsf{wt}(x) \geq (\frac{1}{2} + \varepsilon)n$). Then, by the guarantee of the sampler, for all but $\frac{1}{8}$ of the right vertices in the original graph G, the fraction of neighbors with label 1 of a right vertex is at least $\frac{1}{2} + \varepsilon - \frac{\varepsilon}{2} > \frac{1}{2}$. Thus, all but $\frac{1}{8}$ of the (constant-size) majority circuits located in R output 1. Hence, the fraction of majority circuits that output 0 in R' is at most $\frac{n/8}{n/2} = \frac{1}{4} \leq \frac{1}{2} - \varepsilon$, as desired.

3.1 Supporting Sub-Constant Bias

For sub-constant ε, the sampler technique described above is wasteful, as it requires us to use a sequence of $O(\log n)$ layers with fan-in $O(\varepsilon^{-2})$. For sub-constant ε, this results in a circuit with a super-logarithmic depth. However, we observe that one layer of fan-in $O(\varepsilon^{-2})$ circuits is enough to amplify the bias from ε to 0.4 (rather than just keep the bias at ε). This reduces us to the constant bias case, which can be solved as above with an additional $O(\log n)$-depth.

Thus, in order to obtain an $O(\log n)$-depth circuit on n inputs, that computes majority correctly for inputs with bias at least ε, it is enough to construct an $O(\log n)$-depth circuit with $O(\varepsilon^{-2})$ inputs that computes majority correctly on all inputs.

Using a naive brute-force algorithm, one can efficiently find an optimal-depth circuit on roughly $\log n$ inputs that computes majority. By plugging this circuit into the above scheme, one immediately gets an $O(\log n)$-depth circuit that computes majority on n inputs with bias roughly $\varepsilon = \Omega(\frac{1}{\sqrt{\log n}})$.

We improve on this by using an additional derandomization idea. Specifically, we construct an $O(\log n)$-depth circuit on $2^{O(\sqrt{\log n})}$ inputs, that computes majority (under no assumption on the bias). Thus, we obtain an explicit construction of a circuit that computes majority assuming the bias is at least $\varepsilon = 2^{-O(\sqrt{\log n})}$.

We first describe a randomized construction of an $O(\log m)$-depth circuit on m inputs for majority, where m is set, in hindsight, to $2^{O(\sqrt{\log n})}$. Our construction only uses $O(\log^2 m)$ random bits (compared to $\mathrm{poly}(m)$ random bits used in Valiant's construction). We then show how to derandomize this construction.

Our randomized construction works as follows. Consider an input $x \in \{0,1\}^m$ with bias ε. Suppose that we sample uniformly and independently at random 3 bits of x and compute their majority. It is shown in [Gol11b] that the majority's bias is at least 1.2ε (as long as ε is not too large).

Thus, by placing m majority gates of fan-in 3, and selecting their inputs from x uniformly and independently at random, the output of the m majority gates will have bias of at least 1.1ϵ with overwhelming probability. By composing $O(\log(1/\varepsilon))$ such layers, we can amplify the bias to a constant. Note that this construction uses $O(m \cdot \log m \cdot \log(1/\varepsilon))$ random bits.

To save on the number of random bits used (which is essential for the derandomization step), instead of sampling the inputs of each one of the m gates uniformly at random, we choose them in each layer using a 6-wise independent hash function. While 3-wise independence suffices for the expectation of the bias to be as before, the 6-wise independence guarantees that the outputs of the majority gates in each layer are pairwise independent. Using tail inequalities we show that, with probability $1 - o(1)$, the bias increases in each layer as before.

By composing $O(\log(1/\varepsilon))$ such layers, each of which requires $O(\log m)$ random bits, we obtain a circuit as desired. The total number of random bits used is $O(\log(m) \cdot \log(1/\varepsilon))$, which is bounded by $O(\log^2 m)$. We derandomize the construction by placing all $2^{O(\log^2 m)}$ majority circuits that can be output by the randomized construction and taking the majority vote of these circuits.

Since we have a guarantee that almost all (a $1 - o(1)$ fraction) of the circuits correctly compute majority, it is enough to compute the majority vote at the end using a circuit with $2^{O(\log^2 m)}$ inputs that works for, say, constant bias. Such a circuit, with depth $O(\log^2 m)$, can be constructed in time $2^{O(\log^2 m)}$ by the constant-bias scheme described earlier.

As we set $m = 2^{O(\sqrt{\log n})}$, we get a $\mathrm{poly}(n)$-time uniform construction of an $O(\log n)$-depth circuit on $2^{O(\sqrt{\log n})}$ inputs that computes majority correctly on all inputs. This circuit is then used in the scheme described above.

Threshold formulae from thresholds gates. The scheme described above works also in the more general setting of threshold from thresholds. Indeed, in the paper we present the scheme in the general setting. To apply the scheme in the thresholds setting, one needs to construct a small circuit that computes the required threshold formula, to be used by ShrinkerGenerator. We accomplish this by extending results of [AR63, HM00, BIW10]. See Section 5.

4 Preliminaries for the Complexity Theoretic Results

Let $x \in \{0,1\}^n$. We denote the Hamming weight of x by $\mathsf{wt}(x)$. The normalized Hamming weight is denote by $\mathsf{relwt}(x)$, that is, $\mathsf{relwt}(x) = \mathsf{wt}(x)/n$. We further denote the bias of x by $\mathsf{bias}(x) = |\mathsf{relwt}(x) - 1/2|$. Let $G = (V, E)$ be an undirected graph. The degree of a vertex v is denoted by $d(v)$. For a set $S \subseteq V$ define $d_S(v) = |\{s \in S : sv \in E\}|$.

Circuits and Formulae. Let C be a circuit on n inputs that outputs m bits. For $x \in \{0,1\}^n$, we denote by $C(x) \in \{0,1\}^m$ the output of C when fed with x as input. Let C_1 be a circuit on n inputs that outputs m bits. Let C_2 be a circuit on m inputs that outputs r bits. We denote by $C_2 \circ C_1$ the circuit on n inputs and r outputs that is composed of C_1 and C_2, where the m outputs of C_1 are wired to the m inputs of C_2. Clearly, $C_2 \circ C_1(x) = C_2(C_1(x))$.

The size of a circuit C, denoted by $\mathsf{size}(C)$, is the number of gates in the circuit. The depth, denoted by $\mathsf{depth}(C)$, is the largest number of gates from an input to an output in C.

In this paper we focus on logarithmic-depth formulae composed of constant fan-in threshold gates. Note that such formulae always have polynomial size.

Observation 4.1. *A logarithmic-depth* circuit *composed of constant fan-in gates can be efficiently transformed to a logarithmic-depth* formula *composed of constant fan-in gates, that computes the same function.*

Indeed, one can work his way bottom-up and duplicate every gate with fan-out $k > 1$, together with its sub-formula, to k copies. As the duplication does not change the depth of a gate, the resulting formula has logarithmic depth (and so, due to the constant fan-in, a polynomial size). Thus, in all of our theorems we are satisfied with constructing logarithmic-depth circuits composed of constant fan-in gates.

For integers $0 \le j \le k$ define the threshold function $\mathsf{Th}^k_j : \{0,1\}^k \to \{0,1\}$ as follows. $\mathsf{Th}^k_j(x) = 1$ if and only if $\mathsf{wt}(x) \ge j$. Define $\mathsf{Maj}_{2t+1} = \mathsf{Th}^{2t+1}_{t+1}$. That is, Maj_{2t+1} computes the majority function on $2t+1$ inputs. When the number of variables is clear from the context we omit it from the subscript and write Maj.

From bipartite graphs to circuits. The following notation will be useful for us. Let $G = (L, R, E)$ be a bipartite graph with right-degree $mt + 1$. Define the circuit $C_{G,m,t}$ as follows. $C_{G,m,t}$ has $|L|$ inputs and $|R|$ outputs. With every vertex $r \in R$ associate a threshold circuit Th^{mt+1}_{t+1} in $C_{G,m,t}$. The inputs to this circuit are the $mt+1$ neighbors of r in G. The outputs of $C_{G,m,t}$ are the output of the $|R|$ threshold circuits, one for each vertex in R. When m, t are clear from the context we omit them from the subscript and write C_G.

Samplers. Samplers are key pseudorandom objects we use in our constructions. In this paper we only use a special kind of samplers, known in the literature as *non-adaptive averaging Boolean samplers.* For brevity we call them samplers. For more details we refer the reader to a survey by Goldreich [Gol11a]. It will be useful for us to define samplers in terms of bipartite graphs.

Definition 4.2 (Samplers)**.** *An* $(n, d, \varepsilon, \delta)$-sampler *is a bipartite graph* $\mathsf{Sampler} = (L, R, E)$ *with the following properties:*

- $|L| = |R| = n$.
- $\mathsf{Sampler}$ *has right-degree* d.
- *For every* $S \subseteq L$*, it holds that for at least* $1 - \delta$ *fraction of the vertices* $r \in R$*,*

$$\left| \frac{d_S(r)}{d(r)} - \frac{|S|}{n} \right| \leq \varepsilon.$$

Theorem 6 ([KPS85, Gol11a])**.** *For every* $n \in \mathbb{N}$ *and for every* $\varepsilon = \varepsilon(n) > 0$, $\delta = \delta(n) > 0$*, there exists an* $(n, d, \varepsilon, \delta)$*-sampler* $\mathsf{Sampler}$*, with* $d = O(\varepsilon^{-2} \cdot \delta^{-1})$*. Moreover,* $\mathsf{Sampler}$ *can be constructed in time* $\mathrm{poly}(n, \varepsilon^{-1}, \delta^{-1})$*.*

5 Threshold Formulae from Threshold Gates

In this section we prove Theorem 5. Recall that we assume without loss of generality that $j \geq 2$ and $k \geq 2j - 1$. We start by proving Lemma 4.

5.1 Proof of Lemma 4

To prove Lemma 4 we recall a couple of definitions and prove helpful lemmas.

Definition 5.1 (Q_m functions)**.** *Let* $m \in \mathbb{N}$*. A function* $f : \{0,1\}^n \to \{0,1\}$ *is a* Q_m function *if for every* $x^{(1)}, \ldots, x^{(m)} \in f^{-1}(0)$ *there exists an* $h \in [n]$ *such that* $x_h^{(1)} = \cdots = x_h^{(m)} = 0$.

Lemma 5.2. *Let* $j, k \in \mathbb{N}$*. Every function that can be computed by a formula composed only of* Th_j^k *gates and no constants is a* Q_m *function, where* $m = \left\lfloor \frac{k-1}{j-1} \right\rfloor$*.*

Proof. We prove the lemma by induction on the depth of the formula. Let F be a depth 0 formula composed of Th_j^k gates and without constants. Then, F computes the function $F(x) = x_h$ for some index h. Clearly this is a Q_m function.

Let F be a formula composed of Th_j^k gates without constants, where $\mathsf{depth}(F) > 0$. Let G be the output gate of F. Let $G_1, \ldots, G_k$ be the gates that have their output wired to the inputs of G. By induction, for every $i \in [k]$, the function computed by the sub formula of F with output gate G_i is a Q_m function.

Let x be an input rejected by F. Since G is a Th_j^k gate, at most $j-1$ of the G_i's accept one of the m inputs. Thus, for every m inputs rejected by F, at most $m(j-1) \leq k-1$ of the gates G_i's accept (at least) one of the inputs. Hence, there exists an $\ell \in [k]$ such that G_ℓ rejects all of those m inputs. The proof follows by applying the induction hypothesis on the sub-circuit with output gate G_ℓ. $\square$

Lemma 5.3. *There exists an algorithm A that given constants $j, k \in \mathbb{N}$, where $m = \left\lfloor \frac{k-1}{j-1} \right\rfloor$, and the truth table of a Q_m function $f : \{0,1\}^n \to \{0,1\}$ as input, runs in* $\exp(n)$-*time and generates a formula F on n inputs, with the following properties:*

- *F consists only of Th_j^k gates and no constants.*
- $\mathsf{depth}(F) = O(n)$.
- *$\forall x \in f^{-1}(0)$ it holds that $F(x) = 0$.*

Proof. The algorithm A is recursive. The base case for the recursion is $|f^{-1}(0)| \le m$. In this case, since f is a Q_m function, there exists an $h \in [n]$ such that for every $x \in f^{-1}(0)$ it holds that $x_h = 0$. The algorithm A can find such h in $\exp(n)$-time and return the formula that on input x outputs x_h.

Assume now that $|f^{-1}(0)| \ge m+1$. Partition the set $f^{-1}(0)$ into k sets

$$f^{-1}(0) = A_0 \cup \cdots \cup A_{k-1}$$

of size $\lceil |f^{-1}(0)|/k \rceil$ or $\lfloor |f^{-1}(0)|/k \rfloor$ each, such that $|A_0| \ge |A_1| \ge \cdots \ge |A_{k-1}|$. Let $t \le k-1$ be the maximum integer such that $A_t \ne \emptyset$. Note that $t \ge m+1$ because if there is an empty set among $A_0, \ldots, A_t$ then every non-empty set in the partition has size exactly 1, but their union has size $|f^{-1}(0)| \ge m+1$.

Let $G = (L, R, E)$ be a bipartite graph with $L = \{0, 1, \ldots, k-1\}$, $R = \{0, 1, \ldots, t\}$. A vertex $r \in R$ is connected by edge to the vertices $\{(j-1)r, (j-1)r+1, \ldots, (j-1)r+j-2\} \subseteq L$, where addition is modulo k. By construction, the degree of every right vertex is $j-1$. Moreover, the degree of every left vertex is at least 1 since $(j-1)t + j - 2 \ge k-1$. Indeed,

$$(j-1)t + j - 2 \ge (j-1)(m+1) + j - 2 \ge (j-1)\left(\left\lfloor \frac{k-1}{j-1} \right\rfloor + 1\right) + j - 2 \ge k-1.$$

For $\ell = 0, 1, \ldots, k-1$ define the function $f_\ell : \{0,1\}^n \to \{0,1\}$ to be such that

$$f_\ell^{-1}(0) = f^{-1}(0) \setminus \left(\bigcup_{r : \ell r \in E} A_r \right).$$

For every $\ell = 0, 1, \ldots, k-1$, the algorithm A makes a recursive call on inputs j, k and the truth table of f_ℓ. Denote by $F_0, \ldots, F_{k-1}$ the formulae generated by these recursive calls. Define F to be the formula where the outputs of $F_0, \ldots, F_{k-1}$ are wired to the inputs of a Th_j^k gate, which is the output gate of F. Note that the recursion will eventually get to the base case as the degree of every left vertex in G is at least 1, and so, for every ℓ, $|f_\ell^{-1}(0)|$ is strictly smaller than $|f^{-1}(0)|$.

Let $x \in f^{-1}(0)$. We now show that $F(x) = 0$. Let $r \in \{0, 1, \ldots, t\}$ be the unique integer such that $x \in A_r$. As the degree of r in G is $j-1$, there are exactly $j-1$ functions among $f_0, \ldots, f_{k-1}$ that accept x. Thus, by induction, there are at most $j-1$ formulae among $F_0, \ldots, F_{k-1}$ that accept x. Hence, $F(x) = 0$.

We now analyze the depth and running time of the algorithm. Fix $\ell \in \{0,1,\dots,k-1\}$. If $|f^{-1}(0)| \geq 2k$ it holds that

$$\begin{aligned}|f_\ell^{-1}(0)| &\leq |f^{-1}(0)| - (j-1)\cdot\left\lfloor\frac{|f^{-1}(0)|}{k}\right\rfloor \\ &\leq \left(1-\frac{j-1}{k}\right)\cdot|f^{-1}(0)| + j - 1 \\ &\leq \left(1-\frac{1}{2k}\right)|f^{-1}(0)|.\end{aligned}$$

That is, $f_\ell^{-1}(0)$ has size which is a constant fraction, strictly smaller than 1, of the size of $f^{-1}(0)$, as long as $|f^{-1}(0)| \geq 2k$. Together with the fact that $|f^{-1}(0)| \leq 2^n$, this implies that the depth of the recursion is $O(n) + 2k = O(n)$. We note that the depth of F is exactly the recursion's depth, and so $\mathsf{depth}(F) = O(n)$.

For a function $g : \{0,1\}^n \to \{0,1\}$, let $\mathsf{rt}(g)$ be the running time of A when given as input j, k and the truth table of g. Then,

$$\mathsf{rt}(f) = \sum_{i=0}^{k-1} \mathsf{rt}(f_i) + \exp(n),$$

where the exponential time in n is due to the computation of $A_0, \dots, A_{k-1}$ from $g^{-1}(0)$. Solving the recursion yields $\mathsf{rt}(f) = \exp(n)$. $\square$

Proof of Lemma 4. We first note that Th_{t+1}^{mt+1} is a Q_m function. Let F be the formula generated by the algorithm in Lemma 5.3 on input j, k and the truth table of the function Th_{t+1}^{mt+1}. By Lemma 5.3, every $x \in \{0,1\}^{mt+1}$ such that $\mathsf{wt}(x) \leq t$ is rejected by F.

We now show that every $x \in \{0,1\}^{mt+1}$ such that $\mathsf{wt}(x) \geq t+1$ is accepted by F. Assume for contradiction that there exists $x^{(1)} \in \{0,1\}^{mt+1}$ such that $\mathsf{wt}(x^{(1)}) \geq t+1$ but $x^{(1)}$ is rejected by F. Then, there exist $x^{(2)}, \dots, x^{(m)} \in \{0,1\}^{mt+1}$, each of weight at most t, such that for every $h \in [mt+1]$ there exists an $i \in [m]$ such that $x_h^{(i)} = 1$. On the other hand, since F is composed of Th_j^k gates, by Lemma 5.2, the function computed by F is a Q_m function. This contradicts the existence of such $x^{(1)}$. Thus, $F(x) = \mathsf{Th}_{t+1}^{mt+1}(x)$ for every $x \in \{0,1\}^{mt+1}$.

The depth of F as well as the running time of A follows by Lemma 5.3. $\square$

5.2 Proof of Theorem 5

In this section we prove Theorem 5. A key step in proving Theorem 5 is the following lemma.

Lemma 5.4. *Suppose that there exists an algorithm A' that given $t, j, k \in \mathbb{N}$ as input, where j, k are constants in t, runs in $T'(t)$-time and generates a circuit C' on $mt+1$ inputs, where $m = \left\lfloor\frac{k-1}{j-1}\right\rfloor$, with the following properties:*

- *C' consists only of Th_j^k gates and no constants.*
- *$\mathsf{size}(C') = S'(t)$.*

- $\mathsf{depth}(C') = D'(t)$.
- $\forall x \in \{0,1\}^{mt+1}$, $C'(x) = \mathsf{Th}_{t+1}^{mt+1}(x)$.

Then, there exists an algorithm A that given n, t, j, k as input, where j, k are constants in n, runs in $\mathrm{poly}(n, T'(O(t)))$*-time and generates a circuit C on n inputs, with the following properties:*

- *C consists only of Th_j^k gates and no constants.*
- $\mathsf{size}(C) = O\left(n \cdot S'(O(t))\right)$.
- $\mathsf{depth}(C) = D'(O(t)) + O(\log n)$.
- $\forall x \in \{0,1\}^n$ *such that* $\mathsf{relwt}(x) \geq \frac{1}{m} + \frac{1}{\sqrt{t}}$ *it holds that* $C(x) = 1$.
- $\forall x \in \{0,1\}^n$ *such that* $\mathsf{relwt}(x) \leq \frac{1}{m} - \frac{1}{\sqrt{t}}$ *it holds that* $C(x) = 0$.

To prove Lemma 5.4 we need the following key lemma. Informally, Lemma 5.5 shows how to construct a circuit that shrinks the number of variables to half while maintaining a promise on the weight.

Lemma 5.5. *There exists an algorithm* $\mathsf{ShrinkerGenerator}$ *that given $n \in \mathbb{N}$ as input as well as constants $\varepsilon > 0$ and $j, k \in \mathbb{N}$ such that $0 < \varepsilon \leq \frac{1}{2m}$, where $m = \left\lfloor \frac{k-1}{j-1} \right\rfloor$,* $\mathsf{ShrinkerGenerator}$ *runs in* $\mathrm{poly}(n)$*-time and generates a circuit* $\mathsf{Shrinker}$ *with the following properties:*

- $\mathsf{Shrinker}$ *has n inputs and $\frac{n}{2}$ outputs.*
- $\mathsf{Shrinker}$ *consists only of* Th_j^k *gates and no constants.*
- $\mathsf{size}(\mathsf{Shrinker}) = O(n)$.
- $\mathsf{depth}(\mathsf{Shrinker}) = O(1)$.
- $\forall x \in \{0,1\}^n$ *such that* $\mathsf{relwt}(x) \geq \frac{1}{m} + \varepsilon$ *it holds that* $\mathsf{relwt}(\mathsf{Shrinker}(x)) \geq \frac{1}{m} + \varepsilon$.
- $\forall x \in \{0,1\}^n$ *such that* $\mathsf{relwt}(x) \leq \frac{1}{m} - \varepsilon$ *it holds that* $\mathsf{relwt}(\mathsf{Shrinker}(x)) \leq \frac{1}{m} - \varepsilon$.

Proof. Let $\mathsf{Sampler} = (L, R, E)$ be an $(n, d, \frac{\varepsilon}{2}, \frac{1}{4m})$-sampler. By Theorem 6, $\mathsf{Sampler}$ can be constructed in $\mathrm{poly}(n, \varepsilon^{-1}, m) = \mathrm{poly}(n)$-time with $d = O(\varepsilon^{-2} m) = O(1)$. Consider the circuit C_{Sampler}. Let $t \in \mathbb{N}$ be such that $d = mt + 1$.[13] Since $d = O(1)$, by Lemma 4, the circuit C_d, used in C_{Sampler}, that computes the function Th_{t+1}^{mt+1} can be generated in $O(1)$-time. Clearly, the size and depth of C_d are constants.

The circuit $\mathsf{Shrinker}$ is defined to be the circuit C_{Sampler}, where we arbitrarily choose half of the outputs and discard the rest, together with the respective copies of C_d. By construction, $\mathsf{Shrinker}$ has n inputs and $\frac{n}{2}$ outputs. It consists of Th_j^k gates and no constants. Clearly, $\mathsf{size}(\mathsf{Shrinker}) = \frac{n}{2} \cdot \mathsf{size}(C_d) = O(n)$ and $\mathsf{depth}(\mathsf{Shrinker}) = \mathsf{depth}(C_d) = O(1)$.

[13]For simplicity of presentation we assume $d \equiv 1 \pmod{m}$. This assumption can be met easily.

Let $x \in \{0,1\}^n$ be such that $\mathsf{relwt}(x) \geq \frac{1}{m} + \varepsilon$. Define the set $S_x = \{i \in [n] : x_i = 1\}$. Note that $|S_x| \geq (\frac{1}{m} + \varepsilon)n$. By identifying L with $[n]$, we get by the definition of $\mathsf{Sampler}$, that for at least $1 - \frac{1}{4m}$ fraction of the vertices $r \in R$ it holds that

$$\frac{d_{S_x}(r)}{d(r)} \geq \frac{|S_x|}{n} - \frac{\varepsilon}{2} \geq \frac{1}{m} + \frac{\varepsilon}{2}.$$

Thus, the copy of C_d in C_{Sampler} that is associated with such r gets an input with weight at least

$$\left(\frac{1}{m} + \frac{\varepsilon}{2}\right) \cdot (mt + 1) > t.$$

Thus, the output of C_d that is associated with such r is 1, and so $\mathsf{relwt}(C_{\mathsf{Sampler}}(x)) \geq 1 - \frac{1}{4m}$. Hence, for any subset of size $\frac{n}{2}$ of the outputs of C_{Sampler}, at least $1 - \frac{1}{2m}$ fraction of them are equal to 1. That is,

$$\mathsf{relwt}(\mathsf{Shrinker}(x)) \geq 1 - \frac{1}{2m} \geq \frac{1}{m} + \varepsilon,$$

where the last inequality follows since $m \geq 2$ and $\varepsilon \leq \frac{1}{2m}$.

Let $x \in \{0,1\}^n$ be such that $\mathsf{relwt}(x) \leq \frac{1}{m} - \varepsilon$. In this case $|S_x| \leq (\frac{1}{m} - \varepsilon)n$. By the definition of $\mathsf{Sampler}$, we have that for at least $1 - \frac{1}{4m}$ fraction of the vertices $r \in R$ it holds that

$$\frac{d_{S_x}(r)}{d(r)} \leq \frac{|S_x|}{n} + \frac{\varepsilon}{2} \leq \frac{1}{m} - \frac{\varepsilon}{2}.$$

Thus, the copy of C_d in C_{Sampler} that is associated with such r gets an input with weight at most

$$\left(\frac{1}{m} - \frac{\varepsilon}{2}\right) \cdot (mt + 1) < t + 1.$$

Thus, the output of C_d that is associated with such r is 0, and so $\mathsf{relwt}(C_{\mathsf{Sampler}}(x)) \leq \frac{1}{4m}$. Hence, for any subset of size $\frac{n}{2}$ of the outputs of C_{Sampler}, at most $\frac{1}{2m}$ fraction of them are equal to 1. That is,

$$\mathsf{relwt}(\mathsf{Shrinker}(x)) \leq \frac{1}{2m} \leq \frac{1}{m} - \varepsilon,$$

where the last inequality follows since $\varepsilon \leq \frac{1}{2m}$. □

Proof of Lemma 5.4. Let $\mathsf{Sampler} = (L, R, E)$ be an $(n, d, \frac{1}{2\sqrt{t}}, \frac{1}{2m})$-sampler. By Theorem 6, $\mathsf{Sampler}$ can be constructed in $\mathrm{poly}(n,t,m) = \mathrm{poly}(n,t)$-time with $d = O((2\sqrt{t})^2 \cdot (2m)) = O(t)$. Consider the circuit C_{Sampler}. Let t' be an integer such that $d = mt' + 1$. Note that $t' = O(t)$. By calling A' on input t', j, k, the algorithm A can generate the circuit C' on d inputs, used in C_{Sampler} in $T'(t') = T'(O(t))$-time. Moreover, $\mathsf{size}(C') = S'(t') = S'(O(t))$ and $\mathsf{depth}(C') = D'(t') = D'(O(t))$.

For an integer ℓ, let $\mathsf{Shrinker}_\ell$ be the circuit generated by $\mathsf{ShrinkerGenerator}$ on input $\ell, \frac{1}{2m}, j, k$ (see Lemma 5.5). The algorithm A generates $\mathsf{Shrinker}_\ell$ for $\ell = \frac{n}{2^i}$, where $i = 0, 1, \ldots, \log_2(n) - 1$. The output of A is the circuit

$$C = \mathsf{Shrinker}_2 \circ \cdots \circ \mathsf{Shrinker}_{\frac{n}{4}} \circ \mathsf{Shrinker}_{\frac{n}{2}} \circ \mathsf{Shrinker}_n \circ C_{\mathsf{Sampler}}.$$

Let $x \in \{0,1\}^n$ be such that $\mathsf{relwt}(x) \geq \frac{1}{m} + \frac{1}{\sqrt{t}}$. Define $S_x = \{i \in [n] : x_i = 1\}$. Note that $|S_x| \geq (\frac{1}{m} + \frac{1}{\sqrt{t}}) \cdot n$. By identifying L with $[n]$, and by the definition of $\mathsf{Sampler}$, for at least $1 - \frac{1}{2m}$ fraction of the vertices $r \in R$ it holds that

$$\frac{d_{S_x}(r)}{d(r)} \geq \frac{|S_x|}{n} - \frac{1}{2\sqrt{t}} \geq \frac{1}{m} + \frac{1}{2\sqrt{t}}.$$

Thus, the copy of C' in C_{Sampler} that is associated with such r gets an input with weight at least

$$\left(\frac{1}{m} + \frac{1}{2\sqrt{t}}\right)(mt'+1) > t'.$$

Thus, the output of C' that is associated with such r is 1. Hence,

$$\mathsf{relwt}(C_{\mathsf{Sampler}}(x)) \geq 1 - \frac{1}{2m} \geq \frac{1}{m} + \frac{1}{2m}, \tag{5.1}$$

where the last inequality follows since $m \geq 2$. Define $x^{(1)} = \mathsf{Shrinker}_n(C_{\mathsf{Sampler}}(x)) \in \{0,1\}^{n/2}$. By the definition of $\mathsf{Shrinker}_n$ and by Equation (5.1), we have that $\mathsf{relwt}(x^{(1)}) \geq \frac{1}{m} + \frac{1}{2m}$. Similarly, $x^{(2)} \triangleq \mathsf{Shrinker}_{n/2}(x^{(1)}) \in \{0,1\}^{n/4}$ has weight at least $\frac{1}{m} + \frac{1}{2m}$. Continuing this way we get that $C(x) \in \{0,1\}$ has weight at least $\frac{1}{m} + \frac{1}{2m}$. As $C(x)$ is a single bit it follows that $C(x) = 1$.

Let $x \in \{0,1\}^n$ be such that $\mathsf{relwt}(x) \leq \frac{1}{m} - \frac{1}{\sqrt{t}}$. In this case $|S_x| \leq (\frac{1}{m} - \frac{1}{\sqrt{t}}) \cdot n$. By the definition of $\mathsf{Sampler}$, for at least $1 - \frac{1}{2m}$ fraction of the vertices $r \in R$ it holds that

$$\frac{d_{S_x}(r)}{d(r)} \leq \frac{|S_x|}{n} + \frac{1}{2\sqrt{t}} \leq \frac{1}{m} - \frac{1}{2\sqrt{t}}.$$

Thus, the copy of C' in C_{Sampler} that is associated with such r gets an input with weight at most

$$\left(\frac{1}{m} - \frac{1}{2\sqrt{t}}\right)(mt'+1) < t'+1.$$

Thus, the output of C' that is associated with such r is 0. Hence,

$$\mathsf{relwt}(C_{\mathsf{Sampler}}(x)) \leq \frac{1}{2m} = \frac{1}{m} - \frac{1}{2m}. \tag{5.2}$$

Define $x^{(1)} \triangleq \mathsf{Shrinker}_n(C_{\mathsf{Sampler}}(x)) \in \{0,1\}^{n/2}$. By the definition of $\mathsf{Shrinker}_n$ and by Equation (5.2) we have that $\mathsf{relwt}(x^{(1)}) \leq \frac{1}{m} - \frac{1}{2m}$. Similarly, $x^{(2)} \triangleq \mathsf{Shrinker}_{\frac{n}{2}}(x^{(1)}) \in \{0,1\}^{n/4}$ has weight at most $\frac{1}{m} - \frac{1}{2m}$. Continuing this way we get that $C(x) \in \{0,1\}$ has weight at most $\frac{1}{m} - \frac{1}{2m}$. Since $C(x)$ is a single bit, it follows that $C(x) = 0$.

We now analyze the size and depth of C.

$$\begin{aligned}
\mathsf{size}(C) &= \mathsf{size}(C_{\mathsf{Sampler}}) + \sum_{i=0}^{\log n} \mathsf{size}\left(\mathsf{Shrinker}_{\frac{n}{2^i}}\right) \\
&= n \cdot S'(O(t)) + O\left(\sum_{i=0}^{\log n} \frac{n}{2^i}\right) \\
&= O\left(n \cdot S'(O(t))\right),
\end{aligned}$$

and

$$\mathsf{depth}(C) = \mathsf{depth}\left(C_{\mathsf{Sampler}}\right) + \sum_{i=0}^{\log n} \mathsf{depth}\left(\mathsf{Shrinker}_{\frac{n}{2^i}}\right) = D'(O(t)) + O(\log n).$$

□

Theorem 5 readily follows by Lemma 4 and Lemma 5.4.

Proof of Theorem 5. By calling the algorithm from Lemma 4 on input $t = \log n, j, k$, the algorithm A can generate in $\exp(t) = \mathrm{poly}(n)$-time an $O(t) = O(\log n)$-depth formula on $mt + 1 = O(\log n)$ inputs, composed of Th_j^k gates with no constants, that computes the function Th_{t+1}^{mt+1}. By Lemma 5.4 this implies that in time $\mathrm{poly}(n, \exp(t)) = \mathrm{poly}(n)$, A can generate a circuit C on n inputs that is composed of Th_j^k gates, with the following property. For $x \in \{0,1\}^n$ such that $\mathsf{relwt}(x) \geq \frac{1}{m} + \Omega(\frac{1}{\sqrt{\log n}})$, $C(x) = 1$, and for $x \in \{0,1\}^n$ such that $\mathsf{relwt}(x) \leq \frac{1}{m} - \Omega(\frac{1}{\sqrt{\log n}})$, $C(x) = 0$. Moreover $\mathsf{depth}(C) = O(t + \log n) = O(\log n)$.

By Observation 4.1, the circuit C can be transformed in $\mathrm{poly}(n)$-time to a formula with the same depth, that computes the same function. □

6 Majority Formulae from Majority Gates

In this section we prove Theorem 2 and Theorem 3.

6.1 Proof of Theorem 2

Theorem 2 readily follows by Lemma 5.4 together with the following lemma.

Lemma 6.1. *There exists an algorithm A that given an integer n as input, runs in $\mathrm{poly}(n)$-time and computes a circuit C on $m \triangleq 2^{\sqrt{\log n}}$ inputs, with the following properties:*

- *C consists only of Maj_3 gates and no constants.*
- *$\mathsf{size}(C) = \mathrm{poly}(n)$.*
- *$\mathsf{depth}(C) = O(\log n)$.*
- *$\forall x \in \{0,1\}^m$ it holds that $C(x) = \mathsf{Maj}(x)$.*

To prove Lemma 6.1 we need the following definitions and results on bounded independence distributions. The following well known lemma gives a concentration bound for a sequence of pairwise independent random variables (see, e.g., [Vad11]).

Lemma 6.2. *Let $X_1, \ldots, X_m \in_R [0,1]$ be a sequence of pairwise independent random variables. Let*

$$X = \frac{X_1 + \cdots + X_m}{m},$$

and let $\mu = \mathbb{E}[X]$. Then,

$$\Pr\left[|X - \mu| \geq \varepsilon\right] \leq \frac{1}{m\varepsilon^2}.$$

Definition 6.3 (k-Wise Independent Hash Functions)**.** *Let $n, m, k \in \mathbb{N}$. A family of functions $\mathcal{H} = \{h : [n] \to [m]\}$ is called* k-wise independent *if for every distinct $x_1, \ldots, x_k \in [n]$, the random variables $h(x_1), \ldots, h(x_k)$, where h is sampled uniformly from $\mathcal{H}$, are independent and uniformly distributed in $[m]$.*

Theorem 7. *For every $n, m, k \in \mathbb{N}$ there exists an explicit construction of a k-wise independent family of hash functions $\mathcal{H} = \{h : [n] \to [m]\}$, with size $|\mathcal{H}| = O(\max\{n, m\}^k)$.*

A proof for Theorem 7 can be found in, e.g., [Vad11]. To prove Lemma 6.1 we also need the following fact proved by Goldreich [Gol11b] in his exposition of Valiant's proof. The fact roughly states that the output of a Maj_3 gate, applied to three uniformly sampled entries of a (biased) string x, has a higher bias than the bias of x.

Fact 6.4. *Let n be an odd integer. Let $x \in \{0,1\}^n$ and let $\varepsilon = \mathsf{bias}(x)$. Define*

$$\varepsilon' \triangleq \mathsf{Pr}_{i,j,k\sim[n]}\left[\mathsf{Maj}_3(x_i, x_j, x_k) = \mathsf{Maj}(x)\right] - \frac{1}{2}.$$

Then, $\varepsilon' = 1.5\varepsilon - 2\varepsilon^3$. In particular, if $\varepsilon \leq 0.4$ then $\varepsilon' > 1.15\varepsilon$. Moreover, for every $\varepsilon \in [0, 1/2]$ it holds that $\varepsilon' \geq \varepsilon$.

The following notation will be useful for us. Let $m \in \mathbb{N}$. For a function $f : [3m] \to [m]$, define the bipartite graph $G_f = (L, R, E)$, with $|L| = |R| = m$ as follows. Associate L and R with $[m]$, and connect every vertex $r \in R$ with the vertices $f(3r-2), f(3r-1)$ and $f(3r)$ in L. Note that the right-degree of G_f is 3. We also define $C_f \triangleq C_{G_f}$.

Lemma 6.5. *Let $m \in \mathbb{N}$, and let $\mathcal{H} = \{h : [3m] \to [m]\}$ be the 6-wise independent family of hash functions from Theorem 7. Then, for every $x \in \{0,1\}^m$ such that $\mathsf{bias}(x) \geq \frac{1}{m^{1/4}}$:*

$$\mathsf{Pr}_{h\sim\mathcal{H}}\Big[\mathsf{bias}\left(C_h(x)\right) \leq \min\left\{1.1 \cdot \mathsf{bias}(x), 0.4\right\}\Big] = O\left(\frac{1}{\sqrt{m}}\right).$$

Proof. Fix $x \in \{0,1\}^m$. Since $\mathcal{H}$ is a 6-wise independent family of hash functions, it is in particular 3-wise independent. As the majority gates in C_h have 3 inputs each, we can apply Fact 6.4 and conclude that

$$\mathbb{E}_{h\sim\mathcal{H}}\Big[\mathsf{bias}\left(C_h(x)\right)\Big] \geq \min\left\{1.15 \cdot \mathsf{bias}(x), 0.4\right\}.$$

By the 6-wise independence of $\mathcal{H}$, the outputs of the majority gates in the circuit C_h, where h is sampled uniformly from $\mathcal{H}$, are pairwise independent. Thus, by Lemma 6.2 and since $\mathsf{bias}(x) \geq \frac{1}{m^{1/4}}$,

$$\mathsf{Pr}_{h\sim\mathcal{H}}\Big[\mathsf{bias}\left(C_h(x)\right) < \min\left\{1.1 \cdot \mathsf{bias}(x), 0.4\right\}\Big] \leq \frac{1}{m \cdot (0.05 \cdot \mathsf{bias}(x))^2} = O\left(\frac{1}{\sqrt{m}}\right).$$

□

Proof of Lemma 6.1. We first describe the circuit C and then prove its correctness.

As a first step, the circuit C replicates each one of its m inputs m^3 times to obtain a total of $m' = m^4$ wires. Note that the bias of these wires is at least $\frac{1}{m} = \frac{1}{(m')^{1/4}}$.

Let $\mathcal{H} = \{h : [3m'] \to [m']\}$ be the 6-wise independent family of hash functions from Theorem 7. Recall that $\mathcal{H}$ has size $O((m')^6) = \mathrm{poly}(m)$, and can be constructed in $\mathrm{poly}(m)$-time. Let $h_1, \ldots, h_\ell$ be functions sampled from $\mathcal{H}$ uniformly and independently at random, where ℓ is the smallest integer that satisfies the inequality $1.1^\ell \geq 0.4m$.

Let C' be the circuit generated by the algorithm from Theorem 5 on input $m', j = 2, k = 3$. The circuit C' computes the majority function correctly on inputs with bias at least $o(1)$ and thus certainly when the bias is at least 0.4. Moreover, C' has $\mathrm{poly}(m)$-size, $O(\log m)$-depth, and can be constructed in $\mathrm{poly}(m)$-time. For $h_1, \ldots, h_\ell \in \mathcal{H}$, define the circuit

$$C_{h_1,\ldots,h_\ell} \triangleq C' \circ C_{h_\ell} \circ \cdots \circ C_{h_1}.$$

Let C'' be the circuit generated by the algorithm in Theorem 5 on input $|\mathcal{H}|^\ell, j = 2, k = 3$. Since $|\mathcal{H}|^\ell = m^{O(\log m)} = \mathrm{poly}(n)$, the circuit C'' has $\mathrm{poly}(n)$-size, $O(\log n)$-depth, and can be constructed in $\mathrm{poly}(n)$-time. C'' computes the majority function correctly on inputs with bias at least 0.1 (and, in fact, even for bias at least $o(1)$).

After the replication step, the m' replicated wires are wired as inputs to the circuit $C_{h_1,\ldots,h_\ell}$ for every ℓ-tuple $h_1, \ldots, h_\ell \in \mathcal{H}$. Wire the output of each such circuit to an input of C''. The output of C is defined to be the output of C''. By the above, C can be constructed in $\mathrm{poly}(n)$-time. This implies that $\mathsf{size}(C) = \mathrm{poly}(n)$. Since

$$\mathsf{depth}(C_{h_1,\ldots,h_\ell}) = \mathsf{depth}(C') + \sum_{i=1}^{\ell} \mathsf{depth}(C_{h_i}) = O(\log m) + \ell \cdot O(1) = O(\log m)$$

for every $h_1, \ldots, h_\ell \in \mathcal{H}$, and since $\mathsf{depth}(C'') = O(\log n)$, we have that $\mathsf{depth}(C) = O(\log n)$.

We now show that C computes the majority function. Let $x \in \{0,1\}^m$ and let $x' \in \{0,1\}^{m'}$ be the resulted replicated string. By Lemma 6.5, and by applying a union bound,

$$\mathsf{Pr}_{h_1,\ldots,h_\ell \sim \mathcal{H}} \left[\mathsf{bias}\left(C_{h_\ell} \circ \cdots \circ C_{h_1}(x')\right) < 0.4\right] < \frac{\ell}{\sqrt{m'}} = O\left(\frac{\log m}{m^2}\right) < 0.1.$$

Thus, by the definition of C',

$$\mathsf{Pr}_{h_1,\ldots,h_\ell \sim \mathcal{H}} \left[C_{h_1,\ldots,h_\ell}(x') = \mathsf{Maj}(x)\right] \geq 0.9.$$

That is, at least a 0.9 fraction of the circuits $C_{h_1,\ldots,h_\ell}$ output $\mathsf{Maj}(x)$. This implies that C'', and thus C, outputs $\mathsf{Maj}(x)$.

$\square$

Proof of Theorem 2. By Lemma 6.1, a circuit of $\mathrm{poly}(n)$-size and $O(\log n)$-depth, that computes the majority function on $2^{\sqrt{\log n}}$ inputs can be generated in $\mathrm{poly}(n)$-time. By applying Lemma 5.4 we obtain, in $\mathrm{poly}(n)$-time, an $O(\log n)$-depth circuit that computes the majority under the assumed promise on the bias. The proof then follows by Observation 4.1. $\square$

6.2 Proof of Theorem 3

In this section we prove Theorem 3. We first recall a few definitions and results.

Definition 6.6. *Let $n, s \in \mathbb{N}$. A distribution $\mathcal{R}$ over $\{0,1\}^n$ is* s-pseudorandom *if for every circuit C on n inputs with size at most s, it holds that*

$$|\mathsf{Pr}[C(\mathcal{R}) = 1] - \mathsf{Pr}[C(U_n) = 1]| < 0.1.$$

Definition 6.7. *Let $S : \mathbb{N} \to \mathbb{N}$ be a time-constructible function* [14]*. A function $\mathsf{PRG} : \{0,1\}^* \to \{0,1\}^*$ is an S-pseudorandom generator* [15] *if*

- $\forall r \in \{0,1\}^*$, $|\mathsf{PRG}(r)| = S(|r|)$.
- *There exists an algorithm that, given $r \in \{0,1\}^*$, runs in time $2^{O(|r|)}$ and outputs $\mathsf{PRG}(r)$.*
- $\forall s \in \mathbb{N}$, *the distribution $\mathsf{PRG}(U_s)$ is $(S(s))^5$-pseudorandom.*

In a long line of research initiated by Nisan and Wigderson [NW94] (with some of the ideas, in the context of cryptography, dating back to Yao, Blum and Micali [Yao82b, BM84]), it has been shown that pseudorandom generators can be constructed under hardness assumptions. We state a more recent theorem by Umans [Uma03].

Theorem 8 ([Uma03])**.** *Let $S : \mathbb{N} \to \mathbb{N}$. Given the truth table of a function $f : \{0,1\}^s \to \{0,1\}$ that cannot be computed by a circuit with size at most $S(s)$, there exists an $S(s)^{\Omega(1)}$-pseudorandom generator.*

We also make use of the following theorem.

Theorem 9 ([Val84], see also [Gol11b])**.** *There exists a constant c_1 such that the following holds. For every $n \in \mathbb{N}$ there exists a family of formulae $\mathcal{F}_n = \left\{F_w^{(n)} : w \in \{0,1\}^{n^{c_1}}\right\}$, each on n inputs, such that for every $w \in \{0,1\}^{n^{c_1}}$,*

- *$F_w^{(n)}$ consists only of Maj_3 gates and no constants.*
- $\mathsf{size}(F_w^{(n)}) = O(n^{c_1})$.
- $\mathsf{depth}(F_w^{(n)}) = O(\log n)$,

and

$$\mathsf{Pr}_{w \sim \{0,1\}^{n^{c_1}}}\left[\forall x \in \{0,1\}^n \;\; F_w^{(n)}(x) = \mathsf{Maj}(x)\right] > 1 - 2^{-n}.$$

Moreover, given $n \in \mathbb{N}, w \in \{0,1\}^{n^{c_1}}$, the formula $F_w^{(n)}$ can be constructed in $O(n^{2c_1})$-time.

[14] A function $S : \mathbb{N} \to \mathbb{N}$ is *time constructible* if $\forall s \in \mathbb{N}$ it holds that $S(s) \geq s$, and there exists a Turing machine that on input 1^s outputs $1^{S(s)}$ in $O(S(s))$-time.

[15] Note that this definition refers to pseudorandom generators in the context of computational complexity rather than cryptography.

Proof of Theorem 3. By Theorem 8, the assumption of Theorem 3 implies the existence of a $2^{s/c_2}$-pseudorandom generator PRG, for some constant $c_2 > 1$. Set $s = c_1 c_2 \log_2 n$, where c_1 is the constant from Theorem 9. Note that the PRG outputs n^{c_1} bits.

We first describe the circuit C and then turn to prove its correctness. Let C' be the circuit generated by the algorithm from Theorem 5 on input $n^{c_1 c_2}, j = 2, k = 3$. The circuit C' computes the majority correctly on inputs with bias at least 0.1 (and, in fact, $o(1)$). Moreover, C' has $O(\log n)$-depth, and can be generated in $\mathrm{poly}(n)$-time.

Define the circuit C on n inputs as follows. The n inputs of C are wired to the inputs of the formula $F_w^{(n)}$ for every $w \in \{\mathsf{PRG}(r) : r \in \{0,1\}^s\}$. The output of each such formula is wired to an input of C'. The output of C is defined to be the output of C'.

For every $r \in \{0,1\}^s$, the string $w = \mathsf{PRG}(r)$ can be computed in $\exp(s) = \mathrm{poly}(n)$-time. By Theorem 9, given $n \in \mathbb{N}$ and $w \in \{0,1\}^{n^{c_1}}$, the formula $F_w^{(n)}$ can be generated in $\mathrm{poly}(n)$-time. Since there are $2^s = n^{c_1 c_2}$ such formulae, and since C' can be generated in $\mathrm{poly}(n)$-time, the circuit C can be generated in $\mathrm{poly}(n)$-time. By Theorem 5 and Theorem 9, $\mathsf{depth}(C') = O(\log n)$ and for every $w \in \{0,1\}^{n^{c_1}}$, $\mathsf{depth}(F_w^{(n)}) = O(\log n)$. Hence, $\mathsf{depth}(C) = O(\log n)$, as stated.

We now prove that C computes the majority function. Assume, for contradiction, that there exists $x_0 \in \{0,1\}^n$ such that $C(x_0) \neq \mathsf{Maj}(x_0)$. Then, by the construction of C,

$$\mathsf{Pr}_{r \sim \{0,1\}^s}\left[F^{(n)}_{\mathsf{PRG}(r)}(x_0) = \mathsf{Maj}(x_0)\right] \leq 0.6. \tag{6.1}$$

By Theorem 9, given $n \in \mathbb{N}, w \in \{0,1\}^{n^{c_1}}$, the formula $F_w^{(n)}$ can be constructed in $O(n^{2c_1})$-time. Therefore, there exists a circuit $C_{\text{universal}}$ of size $O(n^{4c_1})$ that has $n^{c_1} + n$ inputs, such that for every $w \in \{0,1\}^{n^{c_1}}, x \in \{0,1\}^n$ it holds that $C_{\text{universal}}(w, x) = F_w^{(n)}(x)$.[16] Consider the circuit C_0 on n^{c_1} inputs defined by hard wiring x_0 as x to $C_{\text{universal}}$. Clearly, $C_0(w) = C_{\text{universal}}(w, x_0)$ for every $w \in \{0,1\}^{n^{c_1}}$. By Equation (6.1),

$$\mathsf{Pr}_{r \sim \{0,1\}^s}\left[C_0(\mathsf{PRG}(r)) = \mathsf{Maj}(x_0)\right] \leq 0.6.$$

On the other hand, by Theorem 9,

$$\mathsf{Pr}_{w \sim \{0,1\}^{n^{c_1}}}\left[C_0(w) = \mathsf{Maj}(x_0)\right] \geq 1 - o(1).$$

This yields a contradiction as C_0, which has size $O(n^{4c_1}) < (n^{c_1} + n)^5$, distinguishes with probability $0.4 - o(1) > 0.1$ a random string from a random output of PRG. $\square$

Weakening the Assumption in Theorem 3. One can show that the assumption in Theorem 3 (i.e., the existence of an $\varepsilon > 0$ such that $\mathsf{E} \triangleq \mathsf{DTIME}(2^{O(n)})$ does not have $2^{\varepsilon n}$-size circuits) can be relaxed to the following assumption: there exists a pseudorandom generator with seed length $O(\log n)$ for read once branching programs of length n and width $O(n)$.[17] The latter assumption is implied by the assumption that appears in Theorem 3.

[16]The quadratic overhead is due to the simulation of an algorithm by a family of circuits; see, e.g., Theorem 6.6 in [AB09].

[17]We do not give here a formal definition for read once branching programs. For more information the reader is referred to [AB09], Chapter 21, Section 6.

The state of the art pseudorandom generator for read once branching programs has seed length $O(\log^2 n)$ [Nis92, INW94]. We stress that this construction is *unconditional* (i.e., no computational assumptions are necessary). We also note that it is possible to use this pseudorandom generator to give an alternative proof for Lemma 6.1, and any advancement in the construction of pseudorandom generators for read once branching programs is a step forward in resolving Conjecture 1. We omit the details in this version of the paper.

7 From Threshold Formulae to Broadcast

In this section, we show a simple way to construct broadcast protocols for n parties from protocols for a constant number of parties based on threshold and also more general formulae. Following the notation of [HM00], we differentiate here between a *processor* which is the entity doing the actual computation and communication and the *player* which is the entity that receives input and produces output. A *party* is the player together with the associated processor.

We will assume throughout that processors can communicate via synchronous secure point-to-point channels. The results and protocols in this section could also be phrased in the more general MPC framework we present later, but we have chosen not to do this, to make the section more self-contained and easier to read. We begin with some basic definitions:

Definition 7.1. *In a* Byzantine Agreement *(BA) protocol, each player P_i has input x_i and output y_i. All honest players must terminate and output the same value, and if $x_i = x$ for all honest P_i then it must be the case that $y_i = x$ for all honest P_i.*

In a Broadcast protocol *some designated player P starts from input x. All honest players must terminate and output the same value y, and if P is honest, it must be the case that $x = y$.*

A 2-cast protocol *is a broadcast protocol for three players.*

It is well known and easy to see that BA implies Broadcast: the broadcaster sends his message to all processors and then we do BA. Conversely, if less than $\frac{n}{2}$ processors are corrupt, then Broadcast implies BA: each processor broadcasts his input and then each processor takes majority decision to get the output. Note that in case the inputs are not bits but come from a larger set, it may happen that that no value is in majority (if the honest processors do not agree on the input to start with). In this case, we adopt in the following the convention that processors output $\perp$.

In the following we want to consider more general adversaries than those that are limited simply by the number of processors they can corrupt. For this we need some definitions:

Definition 7.2. *An* adversary structure *is a family Γ of subsets of the processors such that $A \in \Gamma$, $B \subseteq A$ implies $B \in \Gamma$. An adversary structure is Q_2 (resp., Q_3) if it is the case that no two (resp., no three) subsets in Γ cover the entire processor set. A Q_2-adversary is an adversary that may only corrupt sets of processor belonging to a Q_2 adversary structure. A Q_3-adversary is defined in the same way.*

As an example, an adversary who can corrupt less than $\frac{n}{2}$ of the processors is a special case of a Q_2-adversary.

Assume now we are given a monotone formula F on n inputs. If we let each input variable correspond to a processor, an input string corresponds to a subset of the processors in a natural way, by considering it as the characteristic vector of the subset. The family of subsets that are rejected by F under this correspondence forms an adversary structure because F is monotone. It is not hard to see by induction on the height of F that if it is composed of Th_2^3 gates, the corresponding adversary structure is Q_2, in fact the same arguments shows that any formula composed of Th_j^k gates where $j-1 < k/2$ has a corresponding Q_2 adversary structure (a stronger statement follows by Lemma 5.2). Moreover, for Th_2^4 gates we have a stronger condition, namely they lead to Q_3 adversary structures. Conversely, it follows from [HM00] that for every Q_2 (Q_3) adversary structure Γ, there exists a formula composed of Th_2^3 (Th_2^4) gates that rejects exactly Γ. The formulas obtained this way are not necessarily logarithmic depth, in fact they are very large even for threshold functions.

7.1 Virtual Processors and Formulas

Consider a formula F composed of Th_j^k gates with n inputs. We assume the formula is laid out as a k-ary tree with the output gate on top. As described in the introduction, we assign a virtual processor to the output wire of each gate, and a real processor to each input wire of F. A virtual processor is emulated by the k processors that "sit below him" in the formula.

A virtual processor is defined to be honest if at most $j-1$ of his k emulators are corrupt. An honest virtual processor *holds a value* if all honest processors emulating him agree on that value.

Now assume for a moment that we are given BA for free as a primitive for any group of k processors emulating a virtual processor, where the BA is guaranteed to work if that virtual processor is honest. This immediately implies the following recursively defined protocol for sending a message x from a (virtual or real) processor S to another (virtual or real) processor R:

Message Transfer Protocol (MTP)

1. If S, R are real, send x from S to R.

2. If S is real and R is virtual, S sends x to each processor emulating R using MTP. The emulators do BA to agree on what was received.

3. If S is virtual and R is real, processors emulating S send x to R using MTP, and R takes majority decision to decide what was received.

4. Otherwise (both processors are virtual) each processor emulating S sends x to each processor emulating R using MTP. Each emulator takes majority decision on what he receives, and finally the emulators do BA to agree on what was received.

We will only use this protocol in cases where $j-1 < k/2$. This means that for an honest virtual processor, a majority of his emulators are honest. This and a straightforward inductive argument implies:

Lemma 7.3. *Consider a formula F composed of Th_j^k gates as above, where $j-1 < k/2$. Assume that we are given BA as a primitive for all groups of k processors emulating a virtual processor, where the BA is guaranteed to work if that virtual processor is honest. If S and R are both honest and run the Message Transfer Protocol, R will end up holding x. Even if S is not honest, an honest R always ends up holding some message.*

Under the same assumption as in the above lemma, we can build the following simple protocol for broadcasting a bit, based on formula F composed of Th_j^k gates:

Broadcast Protocol based on F.

1. The broadcaster sends his bit to the virtual processor sitting on top of the formula using MTP.
2. The virtual processor sitting on top of the formula now holds a bit. He sends it to each real processor using MTP.

Theorem 10. *Consider a formula F composed of Th_j^k gates as above, where $j-1 < k/2$. Assume that we are given BA as a primitive for all groups of k processors emulating a virtual processor, where the BA is guaranteed to work if that virtual processor is honest. If the set of corrupt real processors is rejected by F, then the Broadcast Protocol based on F is correct.*

Proof. Since F is built from Th_j^k gates where $j-1 < k/2$, the family of subsets it rejects must be Q_2 as noted above. Hence the set of honest real processors is accepted by F. This means that the virtual processor sitting on top of F is honest. The theorem now follows immediately from Lemma 7.3. □

7.2 Broadcast for less than $n/3$ Corrupted Parties.

As a warm-up we show how to get known results for broadcast in a simple way from the formula based approach.

Corollary 7.4. *The formula based approach implies a broadcast protocol for n parties secure if less than $n(\frac{1}{3} - \frac{1}{\sqrt{\log n}})$ parties are corrupted, and more generally a broadcast protocol secure against any Q_3-adversary.*

Proof. As noted above, there exists a formula composed of Th_2^4 gates that rejects exactly the given adversary structure. This formula satisfies the assumptions in Theorem 10, so all we have to do is to argue that we do BA for 4 processors where at most 1 is corrupt. But this is easy, we can get broadcast and hence BA as follows: the broadcaster P sends his message to all processors using the Message Transfer Protocol, and then each receiver sends to all receivers what he got from P. Finally, the receivers take majority decision to get the output. This construction implies that the BA protocol and the MTP are recursively defined in terms of each other, but this is not a problem, as each recursive call is always to the next lower level, so the recursion eventually "bottoms out". This implies the Q_3-result, and using the formulas constructed in Theorem 5, we get a polynomial time broadcast protocol for the case where at most t processors are corrupted and $t < n(\frac{1}{3} - \frac{1}{\sqrt{\log n}})$. □

This result also follows from the results in [HM00], albeit in a more complicated way. Also, Fitzi and Maurer [FM98] show a broadcast protocol for Q_3-adversaries that is polynomial time in n given access to an oracle that decides membership in the adversary structure.

7.3 From 2-Cast to Broadcast

It was shown by Fitzi and Maurer [FM00] that given 2-cast as a primitive for any group of 3 processors, one can get perfectly secure broadcast for n processors assuming less than $\frac{n}{2}$ are corrupted.

Here, we give an alternative and simple proof of this result, and moreover the same proof trivially extends to show that broadcast is possible in a more general case of a Q_2-adversary. This generalization was previously open.

It is of course easy to do Byzantine agreement (BA) among 3 processors where at least two are honest, if 2-cast is given. So an obvious approach is to do something similar as in the previous corollary: take a formula composed of Th_2^3 gates rejecting exactly the given Q_2 structure and apply Theorem 10. However, to satisfy the assumptions in that result, we need BA, or equivalently 2-cast, among any group of 3 processors, *even virtual processors*. But we are only given 2-cast among real processors.

We therefore need to construct a 2-cast protocol π_{2c} that works for virtual processors, i.e., it is a protocol for 9 processors split in 3 committees of 3 processors each, such that each committee emulates a virtual processor and at least 2 of the 3 virtual processors are honest. There will be a sending virtual processor S and two receivers R_0, R_1. π_{2c} may use 2-cast among any 3-subset of the 9 emulating processors, and must ensure that 1) if S is honest and holds a message (a bit b), then the honest receiver(s) receive b; and 2) if S is not honest, then R_0 and R_1 (who must then be honest) hold the same bit after the protocol[18]. As a first step, we construct a 2-cast protocol π'_{2c}, where the sender is not the entire committee S, but one of the processors in S, here denoted by P_S.

Protocol π'_{2c}

1. P_S 2-casts his bit b to every pair of processors, where one processor is in R_0 and one is in R_1.

2. Each processor in R_0 or R_1 considers all received bits. If the same bit b was received in all 2-casts, store as temporary result b. Else store $\perp$.

3. Each committee R_i ($i = 0, 1$) does BA using the temporary results as inputs. The result will be a bit b_i or $\perp$. The result is reported to the other committee using the message transfer protocol described above.

4. Each processor in R_i computes an output as follows: if your own committee reported b_i, output b_i. If your own committee reported $\perp$ and R_{1-i} reported b_{1-i}, output b_{1-i}. Otherwise (both committees reported $\perp$) output 0.

Lemma 7.5. *Protocol π'_{2c} implements correctly 2-cast from P_S to R_0 and R_1.*

[18] Note that we cannot rely on the result by Fitzi and Maurer to get such a protocol, since we do not have honest majority: it may be the case that only 4 of the 9 processors are honest.

Proof. If the sender P_S is honest and sends b, then if the committee R_i is honest, then the (at least 2) honest processors in R_i will only see b, therefore R_i will report b and honest processors from R_i will output b as they should no matter what happens in R_{1-i}.

If P_S is corrupt, we can assume that both R_0 and R_1 are honest and we must show that they output the same bit. Clearly, if at least one of R_0 and R_1 report $\perp$ or they both report the same bit b, then R_0 and R_1 output the same bit. We now show that the only remaining case where they report different bits cannot occur: assume without loss of generality that R_0 reports 0. In order for this to happen, at least one of the honest processors in R_0 must have 0 as temporary result. But this means (by the 2-casts we used in the first step) that all honest processors in R_1 see at least one 0. Therefore none of them can have 1 as temporary result, and R_1 therefore cannot report 1. □

To get the protocol π_{2c}, we simply run π'_{2c} 3 times with each processor in S playing the role of P_S, and processors in R_0, R_1 take majority decision on the bit to output. The properties we wanted for π_{2c} now follow immediately from the above lemma.

This and Theorem 10 now immediately implies

Corollary 7.6. *Given 2-cast among any group of 3 parties, there exists a broadcast protocol for n parties secure against any Q_2-adversary.*

Note that our results on construction of formulas for the majority function (Theorems 3, 2) together with the construction behind the corollary immediately gives polynomial time (in n) broadcast protocols for at most t corrupted processors. We can use a randomized construction or a construction conditioned on a complexity assumption to get the optimal result $t < \frac{n}{2}$, matching the result from [FM00]. Or we can use the (unconditional) explicit construction that gives us $t < n(\frac{1}{2} - 2^{-O(\sqrt{\log n})})$.

8 The Multiparty Computation Framework

In Section 8.1 we give a general overview of the player emulation technique and in Section 8.2 we give a formal definition of the MPC framework that we will use (based on the [HM00] MPC framework).

8.1 The Player Emulation Technique

The formulas constructed in Section 5 and Section 6 will be used to construct efficient multiparty protocols via the "player emulation" technique from [HM00]. Variants of this technique, also referred to as player *virtualization* or *simulation*, were used for different purposes in several other works (e.g. [Bra87, Cha89, HIKN08, DIK$^+$08, IPS08, LRM10, LOP11]). While implementing player emulation in the passive security model is quite straightforward, in the active security model it requires more care. In the following we give more details on the implementation of this technique.

Recall that in a single player emulation step, the role of a party τ participating in a protocol Π is replaced by a secure protocol π which involves a small set of parties $v_1, \ldots, v_k$, along with the parties of Π. We will typically let $k = 3$ (resp., $k = 4$) in the case of security against a passive (resp., active) adversary, and let π be a protocol which remains secure as long as at most one of the emulating parties v_i is corrupted.

Furthermore, the total computational complexity of all parties in π (which is typically cast in some algebraic computation model) is only bigger by a constant factor than that of the emulated party τ in Π. As explained in the Introduction, a logarithmic-depth threshold formula defines a sequence of such player emulation steps which result in transforming an atomic protocol π for a constant number of parties into an efficient n-party protocol which tolerates an optimal or near-optimal fraction of corrupted parties.

The application of the player emulation technique in [HM00] is formulated in a specialized framework for secure MPC and is restricted to the protocol compiler of BGW [BGW88].[19] However, the technique is quite insensitive to many of these details and can be applied with other protocols and notions of security from the literature.

A conceptually simple way for implementing a player emulation step is by viewing the role of τ in Π as a *reactive* ideal functionality, which interacts with the parties in Π (receiving incoming messages as inputs and delivering outgoing messages as outputs), and maintains a state information during this interaction. The protocol π emulating τ then needs to realize the corresponding functionality using the emulating parties v_i instead of τ. Note that protocol π does not only involve the players emulating τ. It also specifies how players communicating with τ should translate their messages into whatever format π uses.[20]

The protocol π can satisfy any composable notion of security that applies to reactive functionalities, namely one which ensures that π can be securely used as a substitute for τ in an arbitrary execution environment if at most a single v_i is corrupted. The protocols π we use in this work all satisfy the standard notion of UC-security from [Can01], which suffices for this purpose.[21]

Alternatively, it is possible to implement a player emulation step by only relying on protocols for secure function evaluation which satisfy the standard definitions of standalone security [Can00, Gol04]. The idea is to first ensure that only a single message is sent in each round of Π, and then implement a round in which τ interacts with party P by a protocol involving P and the emulating parties v_i. The functionality realized by such a protocol is determined by the choice of a concrete (robust) secret sharing scheme which is used to distribute the state of τ between the emulating parties.

8.2 The [HM00] MPC Framework

We consider n players that wish to perform some computational task together. We focus only on perfect security and, as usual in this context, we assume secure point-to-point communication channels between every two processors. We introduce an abstract framework in which the specific operations that can be performed by the individual

[19] In Section 8.2 we use a slightly stronger variant of the model from [HM00] which supports player emulation without any further requirements.

[20] Alternatively, if the communication channels are modeled as an ideal functionality, one can extend the definition of this functionality so it will do the translation, and then in a final step implement the translation. This leads in some cases to a slightly simpler protocol in the end.

[21] In particular, all these protocols are perfectly secure with a straight-line black-box simulator, which was shown in [KLR10] to imply UC-security in the case of secure function evaluation. We note that while standard UC-security is cast in an asynchronous network model and does not guarantee output delivery, it can be extended to capture synchronous protocols which guarantee output delivery (cf. [CDN12, Chapter 4]).

processors remains undefined. This allows us to obtain generic results that are applied in subsequent sections to concrete multiparty computation models that are instantiations of our abstract framework. Our definitions are based on [HM00], except that we make an additional "locality" requirement which is implicitly used in [HM00].

As in [HM00], we distinguish between a player and the corresponding processor. A *player* is the entity that receives input and produces output and the *processor* is in charge of executing operations and communicating. A *party* is a player together with the associated processor.

We assume a global variable space $\mathcal{X}$. A variable $x \in \mathcal{X}$ can take value from some fixed finite set of values $\mathcal{V}$. We associate with each variable a set of processors that know the value of the variable. We assume that variables are only used in a write-once manner and therefore, transmitting a message between two processors simply entails of adding a variable that is in the sending processor's view to the receiving processor's view.

A *multiparty computation model* (or *MPC model*) defines the set of values $\mathcal{V}$ that may be assigned to variables and the different operators (and respective operands) that the processors may use. The MPC model may also specify that some of these variables have predetermined values and are treated as constants. Additionally the MPC model specifies the power of the adversary (i.e., whether it is passive or active, see Section 8.2.1). All the following definitions are with respect to some fixed MPC model $\mathcal{M}$ which will always be clear from the context.

A *protocol* π among a set P of processors, that involves variables from a variable space $\mathcal{X}$, is a sequence of statements. Each statement may be of the following forms:

1. An input statement $input(p_i, x)$ instructs the processor $p_i \in P$ to read a value from its input tape and to assign the value to its local variable $x \in \mathcal{X}$.

2. A transmit statement $transmit(p_1, p_2, x)$ instructs the processor $p_1 \in P$ to send the value of the variable $x \in \mathcal{X}$ (that is in its view) to the processor $p_2 \in P$. This simply means that x is added to the view of p_2.

3. An output statement $output(p, x)$ instructs the processor $p \in P$ to output the value of the variable $x \in \mathcal{X}$ to its associated processor. We define by $\mathsf{output}(p)$ the list of all values that p has output throughout the execution of the protocol and by $\mathsf{output}_i(p)$ the output of p before the execution of the i-th statement of the protocol.

4. A computation statement $comp(p, op, X, x)$ instructs the processor p to perform the atomic operation op on the variables $X \subseteq \mathcal{X}$ (that are in its view) and to assign the result to the variable $x \in \mathcal{X}$. This may include assigning a random value. The specification of the operation as well as its operands are a part of the MPC model $\mathcal{M}$.

A *multiparty computation specification* (or simply called *specification*) formally specifies the task to be accomplished by the processors. Intuitively, a specification specifies the cooperation in an ideal environment which includes, in addition to the n processors, a trusted processor. Formally, a specification is a pair (π, τ) consisting of a protocol π among a set P_0 of processors, and the name of a virtual processor $\tau \in P_0$. The protocol π of the specification is also called the ideal protocol. We assume that τ is never involved in *input* and *output* statements.

As an example, consider a specification described by a protocol in which all the parties first read their input, send their inputs to the trusted processor τ who computes some function of their input and transmits a result of this computation to each processor. Each processor outputs its own result. We note that this type of specification is known as secure function evaluation.

A *multiparty protocol generator* G for the set P_G of processors is a polynomial-time algorithm that takes as input a multiparty computation specification (π_0, τ) involving processors from a set P_0 and returns a protocol π for the processors $(P_0 \backslash \{\tau\}) \cup P_G$. A *statement index function* for a specification (π_0, τ) and protocol π is a strictly monotone function f,

$$f : \{1, \ldots, |\pi_0| + 1\} \to \{1, \ldots, |\pi| + 1\}$$

where $f(1) = 1$ and $f(|\pi_0|+1) = |\pi|+1$. We say that a protocol generator is *local* if each statement α in π_0 is mapped to a sequence of statements in π such that the mapping does not depend on variables that do not appear in the statement α. All protocol generators considered in this work are local.

Intuitively, a protocol generator G simulates the virtual trusted processor τ by a multiparty computation protocol among the processors in P_G. Each statement of the ideal protocol π_0 is expanded into a sequence of statements, and all these sequences are concatenated to the resulting protocol π. The statement index function f specifies how statements in π_0 are expanded into subprotocol in π. Thus, each index i maps a statement in π_0 to the index $f(i)$ of the first statement in the corresponding subprotocol in π.

We consider different levels of explicitness of the protocol generator. If the protocol generator can be implemented by a polynomial-time Turing Machine, we say that it is *explicit.* If it can be implemented by a probabilistic polynomial-time Turing Machine that only fails with probability that is exponentially vanishing in the number of players then we say that it is a *randomized construction.* We will be mainly interested in protocol generators that only have *blackbox* access to an algebraic structure $\mathcal{V}$. This can be modeled by assuming that each element in $\mathcal{V}$ is given some adversarially chosen identifier.

For an integer $k \in \mathbb{N}$, a k-processor protocol generator is a protocol generator that supports specification involving at most k processors other than the trusted processor τ.

8.2.1 Security Notions for Multiparty Protocols

Let π be a protocol for the set P of processors. Our framework supports adversaries that may be either active or passive (whether the adversary is passive or active is defined as part of the MPC model).

A *passive* adversary A for the protocol π that corrupts the processors Z_A is a (probabilistic) strategy. After each statement of the protocol π, the passive adversary A may read the variables in the views of the corrupted processors Z_A and can extend its own current view based on these values. It may also do arbitrary computation over its own view. The adversary is computationally unbounded.

An *active adversary* A for the protocol π is a passive adversary that may, in addition, take complete control over the corrupted processors Z_A.

Let A be an adversary and (π_0, τ) be a specification for the set of processors P_0. We say that the protocol π *A-securely computes the specification* (π_0, τ) if there exists a statement index function $f : \{1, \ldots, |\pi_0| + 1\} \to \{1, \ldots, |\pi| + 1\}$ and an adversary A_0 for the ideal protocol π_0 with $Z_{A_0} = Z_A \cap (P_0 \backslash \{\tau\})$ that satisfy the following property. For all inputs and for every $i = 1, \ldots, \pi_0 + 1$, the joint distribution of A_0's view with the output of $\mathsf{output}_i(p)$ for all non-corrupted processors $p \in (P_0 \backslash \{\tau\} \backslash Z_{A_0})$ before the i-th statement of the ideal protocol π_0 (with the adversary A_0 present) is equal to the joint distribution of A's view and the views $v_{f(i)}(p)$ of all non-corrupted processors $p \in (P_0 \backslash \{\tau\} \backslash Z_{A_0})$ before the $f(i)$-th statement of the real protocol π (with the adversary A present). Moreover, the complexity of A_0 must be polynomial in the complexity of A.

The adversary A_0 can be thought of as a kind of simulator for A.

A *structure* $\mathcal{Z} \subseteq 2^P$ is a monotone set of subsets of the processors P where by monotone we mean that it is closed to taking subsets. For a structure $\mathcal{Z}$ and a specification (π_0, τ), we say that a protocol π *$\mathcal{Z}$-securely computes* (π_0, τ) if for every adversary A such that $\mathcal{Z}_A \in \mathcal{Z}$, it holds that π A-securely computes (π_0, τ).

A protocol generator G for the set of processors P is A-secure if, for every specification, the protocol that results by applying G to this specification A-securely computes the specification. For a structure $\mathcal{Z} \subseteq 2^P$, a protocol generator G for the set P of processors is $\mathcal{Z}$-secure if, for every adversary A such that $\mathcal{Z}_A \cap P \in \mathcal{Z}$, the protocol generator is A-secure. (See [HM00] for further details.)

8.2.2 Remapping and Simulating Processors

Let P and P' be sets of processors. A processor mapping $\sigma : P \to P'$ is a surjective function from P onto P'. If π is a protocol and $\sigma : P \to P'$ is a processor mapping then the mapped protocol $\sigma(\pi)$ is the same protocol, where each statement involving a processor in $p \in P$ is replaced by the processor $\sigma(p)$.

For a specification (π_0, τ) with $\tau \notin P$, the mapped specification $\sigma(\pi_0, \tau)$ is defined as $(\sigma(\pi_0), \tau)$.

The inverse processor mapping σ^{-1} of a processor mapping is defined by

$$\sigma^{-1} : P' \to 2^P$$

where $\sigma^{-1}(p') = \{p \in P : \sigma(p) = p'\}$. For a set of processors P, we define $\sigma(P) = \cup_{p \in P} \{\sigma(p)\}$ and $\sigma^{-1}(P) = \cup_{p \in P} \sigma^{-1}(p)$.

For a structure $\mathcal{Z}$ for the set P of processors and a processor mapping $\sigma : P \to P'$, the mapped structure is $\sigma(\mathcal{Z}) = \{Z \subseteq P' : \sigma^{-1}(Z) \in \mathcal{Z}\}$.

Lemma 8.1 (Processor Remapping Lemma, [HM00])**.** *Given a protocol π for the set P of processors that $\mathcal{Z}$-securely computes the specification (π, τ), and some processor mapping σ, then $\sigma(\pi)$ is a protocol for the set $\sigma(P)$ of processors that $\sigma(\mathcal{Z})$-securely computes the specification $\sigma(\pi_0, \tau)$.*

Consider a multiparty protocol π among the set P of processors and a protocol generator G for the set P_G of processors. To simulate a virtual processor $p \in P$ in π applying the protocol generator G means to consider this processor p as a trusted processor and to have this processor simulated by a subprotocol among the processors in P_G, according to G. More precisely, the specification (π, p) is used as input for the protocol generator G. We note that only processors that do not have *input* and *output* statements are simulated.

The following theorem of [HM00] show that a processor in an MPC protocol can be simulated by other processors.[22]

Theorem 11 (Processor Simulation Theorem, adapted from [HM00])**.** *Let π be a protocol in the MPC-model $\mathcal{M}$ among the set P of processors that $\mathcal{Z}$-securely computes a specification (π_0, τ), and let $G_1, \dots, G_k$ be $\mathcal{Z}_1, \dots, \mathcal{Z}_k$-secure local protocol generators for the processor sets $P_1, \dots, P_k$, respectively. Assume that in π the k processors $p_{r_1}, \dots, p_{r_k} \in P$ (which have no input or output statements) are simultaneously simulated by subprotocols applying the protocol generators $G_1, \dots, G_k$ respectively. Then the resulting multiparty protocol π^* (also in the MPC-model $\mathcal{M}$) is for the set P^* of processors and $\mathcal{Z}^*$-securely computes the specification (π_0, τ), where*

$$P^* = (P \backslash R) \cup \bigcup_{i=1}^{k} P_i,$$

$$\mathcal{Z}^* = \left\{ Z \subseteq P^* : \Big(\big(Z \cap (P \backslash R)\big) \cup \Big\{ p_{r_i} \in R : Z \cap P_i \notin \mathcal{Z}_i \Big\} \Big) \in \mathcal{Z} \right\},$$

and $R = \{p_{r_1}, \dots, p_{r_k}\}$ is the set of replaced processors.

9 From Threshold Formulae to Secure Multiparty Computation

In this section we show how to use logarithmic depth threshold formulae to obtain an efficient generic reduction from multiparty MPC protocols to MPC protocols for a constant number of parties. Before proceeding to the statement and proof of Lemma 9.1 which captures our approach, we recall some standard notations that will be used in this section.

Notation. For a set $I \subseteq [n]$, we denote by 1_I the n-bit indicator vector $(b_1, \dots, b_n) \in \{0,1\}^n$ where $b_i = 1$ if $i \in I$ and $b_i = 0$ otherwise. Let F be a depth d formula. We say that the output wire of F has depth 0. For any other wire, we say that it has depth i if it is an input to a gate whose output wire has depth $i-1$. For a wire w, that is not an input wire, we define by $\mathsf{children}(w)$ the input wires of the gate of which w is an output wire. For a subset of *input* wires S, we define $F(S)$ we be the value obtained by evaluating F when the input wires of S have value 1 and the other input wires have value 0.

Lemma 9.1. *Let $j < k < n$ be integers and suppose that F is a formula on n inputs, which uses only Th_j^k gates and uses no constants. If there exists an explicit and local k-processor protocol generator for the MPC model $\mathcal{M}$ that is secure against the structure $\{Z \subseteq \{p_1, \dots, p_k\} : |Z| < j\}$ then there exists an explicit $\mathcal{Z}_F$-secure n-processor protocol generator also in the MPC model $\mathcal{M}$ for a set $P = \{p_1, \dots, p_n\}$ of n processors where,*

$$\mathcal{Z}_F = \left\{ \{p_{i_1}, p_{i_2}, \dots, p_{i_k}\} : k \le n \text{ and } F(1_{\{i_1, \dots, i_k\}}) = 0 \right\}.$$

[22]The actual theorem in [HM00] restricts the simulating protocol generators to [BGW88] protocol generators. However, as stated in [HM00, Footnote 16], the only property of the [BGW88] protocol generators that they use is that they are *local* (see Section 8).

Given a specification (π, τ), the protocol generator outputs a protocol π' that $\mathcal{Z}_F$-securely computes (π, τ) such that the $|\pi'| = c^{\mathsf{depth}(F)} \cdot |\pi|$ for a suitable constant c.

Proof. Let G_k be the k-processor protocol generator (that is secure against adversaries that control less than j processors). Given a specification (π, τ) for the processor set $\{p_1, \dots, p_n, \tau\}$, our protocol generator computes a sequence of protocols $\pi_0, \dots, \pi_d, \pi_{d+1}$ and finally outputs the protocol π_{d+1}. Each protocol π_i involves a set of processors P_i (which includes $p_1, \dots, p_n$ but may include additional processors) and $\mathcal{Z}_i$-securely computes (π, τ), for a specific adversary structure $\mathcal{Z}_i$. Note that the protocol generator only outputs the final protocol π_{d+1}, which only involves the processors $P \triangleq \{p_1, \dots, p_n\}$. The main part of the proof will be to show that π_{d+1} is indeed secure with respect to the adversary structure $\mathcal{Z}_F$. In other words, that $\mathcal{Z}_{d+1} = \mathcal{Z}_F$.

The protocols $\pi_1, \dots, \pi_d$ are constructed recursively, whereas π_0 is described directly and π_{d+1} is a simple transformation of π_d. For $i \in \{0, \dots, d\}$, the protocol π_i involves the processors $p_1, \dots, p_n$ (which we call real processors) as well as additional processors (which we call virtual processors). Each protocol π_i is constructed by simulating the virtual processors in π_{i-1} by other virtual processors. Only in the very last step (i.e., the $d+1$-th protocol), all the remaining virtual processors are simulated by real processors.

We associate each wire of F with a (virtual) processor. For a wire w, we abuse notation and use w to denote both the wire and the associated processor. It will be clear from the context whether we think of w as a wire or as the associated processor.

We denote by W_i the set of wires that are either of depth (exactly) i or are *input wires* and are of depth no larger than i. Note that W_0 contains only the output wire of F whereas W_d includes all of the wires of F.

We proceed to describe the protocols $\pi_0, \dots, \pi_d$. For every $i \in \{0, \dots, d\}$, the protocol π_i will involve the (virtual) processors associated with wires in W_i as well as the (real) processors P. That is, $P_i = P \cup W_i$.

We first describe the protocol π_0. As noted above, π_0 involves the processors $P_0 = \{p_1, \dots, p_n, w_0\}$ where w_0 is the processor associated with the output wire (i.e., the single wire of depth 0). The protocol π_0 is simply the protocol π where τ (the trusted processor) is replaced with w_0. Formally, let σ be the processor mapping that maps w_0 to τ. Then $\pi_0 = \sigma(\pi)$. Let $\mathcal{Z}_0 = \{Z \subseteq P_0 \colon Z \subseteq P\}$. Since w_0 acts as a trusted processor in $\mathcal{Z}_0$, the protocol π_0 trivially $\mathcal{Z}_0$-privately computes the specification (π, τ).

For $i \in \{1, \dots, d\}$, we define π_i recursively, based on π_{i-1}. Let $R_{i-1} \subseteq W_{i-1}$ be the subset of wires of depth $i-1$ that are not input wires. The protocol π_i is constructed using Theorem 11 where the processors R_{i-1} are (simultaneously) replaced by their children. Specifically, every processor $w \in R_{i-1}$ is replaced by its own children $\mathsf{children}(w) \subseteq W_i$ using the k-processor local protocol generator G_k. By Theorem 11, the resulting protocol π_i, $\mathcal{Z}_i$-privately computes the specification (π, τ) for

$$\mathcal{Z}_i \triangleq \left\{ Z \subseteq P_i : \big(Z \cap (P_{i-1} \setminus R_{i-1})\big) \ \cup \ \left\{ w \in R_{i-1} : \left| Z \cap \mathsf{children}(w) \right| \geq j \right\} \in \mathcal{Z}_{i-1} \right\}.$$

The final protocol π_{d+1} is obtained by renaming every virtual processor w (corresponding to an input wire w) by a real processor. More specifically, if w is connected to the i-th input variable, then w is simulated by p_i. Formally, let ϕ be a processor mapping that maps every input wire w that is connected to the i-th input variable to p_i. We define $\pi_{d+1} = \phi(\pi_d)$.

To prove that $\mathcal{Z}_{d+1} = \mathcal{Z}_F$, we introduce the notion of a suffix of a formula. We denote by F_i the i-deep suffix of F. That is, F_i is the formula that contains all the wires in $\cup_{j \le i} W_j$. Note that the gates at depth i are not included. Also, note that the wires W_i are included in F_i but are not the output of any gate (in F_i). The wires W_i are used as the input wires of F_i. It is easy to see that F_0 is a trivial formula taking 1 input to 1 output. We also point out that F_d is not (necessarily) equal to F since different input wires of F_d may be wired to the same input *variable* of F.

Claim 9.2. *For every $i \in \{0, \dots, d\}$ it holds that $Z \in \mathcal{Z}_i$ if and only if $F_i(Z \setminus P) = 0$ where F_i is the i-deep suffix of F.*

Proof. We prove the claim by induction. For $i = 0$, by the definition of $\mathcal{Z}_0$, it holds that $Z \in \mathcal{Z}_0$ if and only if $Z \subseteq P$. Note that the formula F_0 is just the trivial formula composed of a single wire that is both the input and output wire. Hence, $Z \in \mathcal{Z}_0$ if and only if $F_0(Z \setminus P) = 0$.

Let $i \in \{1, \dots, d\}$ and suppose that the claim holds for $i - 1$. Let $Z \in P_i$ and let

$$Z' \triangleq \big(Z \cap (P_{i-1} \setminus R_{i-1})\big) \;\cup\; \Big\{ w \in R_{i-1} : \Big| Z \cap \mathsf{children}(w) \Big| \ge j \Big\}. \tag{9.1}$$

By the definition of $\mathcal{Z}_i$, it holds that $Z \in \mathcal{Z}_i$ if and only if $Z' \in \mathcal{Z}_{i-1}$. By the inductive hypothesis, $Z' \in \mathcal{Z}_{i-1}$ if and only if $F_{i-1}(Z' \setminus P) = 0$. Thus, it suffices to show that $F_i(Z \setminus P) = F_{i-1}(Z' \setminus P)$.

Consider the evaluation of the formula F_i on input $Z \setminus P$. That is, the evaluation of F_i when the only input wires of F_i that have value 1 are those in $Z \setminus P$. Consider the values of the wires W_{i-1} during the evaluation of the formula. We show that the wires of W_{i-1} that have value 1 are exactly those in Z'. Fix a wire $w \in W_{i-1}$. We separate into two cases:

1. Suppose that $w \notin R_{i-1}$ (i.e., w is an input wire).
   ($\Rightarrow$) If w has a value of 1 then $w \in Z$ (since only input wires in Z have a value of 1). Thus, by definition of Z' and since $w \notin R_{i-1}$, it holds that $w \in Z'$.

   ($\Leftarrow$) On the other hand, if $w \in Z'$, then since $w \notin R_{i-1}$, it holds that $w \in Z$ and therefore has value 1.

2. Suppose that $w \in R_{i-1}$.
   ($\Rightarrow$) If w has value 1 then, since w is the output of a Th_j^k gate, at least j of its children have value 1 and in particular at least j of w's children are in Z. Thus, by definition of Z', it holds that $w \in Z'$.

   ($\Leftarrow$) On the other hand if $w \in Z'$ then, by the definition of Z', at least j of w's children are in Z. The children of w are input wires and therefore have value 1. Since w's value is computed as the threshold of at least j of its children, w has value 1.

Thus, $F_i(Z \setminus P) = F_{i-1}(Z' \setminus P)$ and the claim follows. □

By Claim 9.2, $Z \in \mathcal{Z}_d$ if and only if $F_d(Z \backslash P) = 0$. By Lemma 8.1,

$$\mathcal{Z}_{d+1} = \phi(\mathcal{Z}_d) = \left\{ Z \subseteq P : \phi^{-1}(Z) \in \mathcal{Z}_d \right\}.$$

Claim 9.3. *For $Z \subseteq P$ it holds that $Z \in \mathcal{Z}_{d+1}$ if and only if $F(Z) = 0$.*

Proof. ($\Rightarrow$) Suppose that $Z \in \mathcal{Z}_{d+1}$, then $Z \subseteq P$ and $\phi^{-1}(Z) \in \mathcal{Z}_d$. In particular $F_d(\phi^{-1}(Z) \backslash P) = 0$. Now consider the evaluation of F on input 1_Z. An input wire w that is connected to the i-th input variable has value 1 if and only if $p_i \in Z$. Thus, the input wires correspond exactly to $\phi^{-1}(Z) \backslash P$. Since the formula proceeds by evaluating F_d on the input wires we have $F(1_Z) = 0$.

($\Leftarrow$) If $F(Z) = 0$, then in particular $F_d(\phi^{-1}(Z)) = 0$. Hence, $\phi^{-1}(Z) \in \mathcal{Z}_d$ and so $Z \in \mathcal{Z}_{d+1}$. □

We conclude that the protocol π_{d+1} $\mathcal{Z}_F$-securely computes the specification (π, τ). Observe that $|\pi_0| = |\pi|$ and that the Processor Simulation Theorem (Theorem 11) replaces every statement in π_{i-1} by a constant number of statements in π_i. Therefore, $|\pi_d| = c^{\mathsf{depth}(F)}|\pi|$ for some suitable constant c. The protocol π_{d+1} has the same length as π_d since it is constructed by replacing every statement in π_d with a single statement. Therefore, the protocol generator outputs a protocol of length $c^{\mathsf{depth}(F)}|\pi|$. □

10 Secure MPC over Blackbox Rings

In this section we consider multiparty computation over any commutative ring $(R, +, *)$. The processors are only given blackbox access to the ring. This model of secure computation was first considered by Cramer *et al.*[CFIK03] and it generalizes the classical model of [BGW88], where the computation is over a finite field. We note that the classical results of [BGW88] for passive and active security do not extend to this model as they require field operations[23]. We present a protocol generator based on the threshold formula to MPC approach introduced in Section 9. We stress that our protocols use the ring in an entirely blackbox manner, are fairly simple and do not rely on any nontrivial facts from algebra (as in [CFIK03]).

Formally, we define an MPC model which we call the *Ring-MPC model* as follows. Every variable $x \in \mathcal{X}$ may take values in the ring R. We allow processors to compute addition and multiplication over the ring. That is, protocols in the Ring-MPC model support the operator $+$ (resp., $*$), which takes two operands and returns their sum (resp., product). Additionally, the processors can sample a random ring element and have access to the constant -1 (i.e., the additive inverse of the multiplicative neutral element of the ring R).

10.1 The Passive Model

Our approach to constructing secure multiparty protocol generators is to use Lemma 9.1 to reduce the problem to that of constructing protocols for a constant number of processors. For the latter, we use Maurer's [Mau06] simple and elegant protocol generator. We

[23]Specifically, the ability to find multiplicative inverses is used by Shamir's secret sharing scheme [Sha79].

note that Maurer's protocol works over any ring and uses the ring in a blackbox manner. A downside of Maurer's protocol generator is that it produces protocols of length exponential in the threshold of tolerated adversaries. However, this does not concern us since we only need to use the base protocol for a constant number of processors.

In fact, instead of using Maurer's protocol generator we could also use the classical [BGW88] protocol. We choose to use Maurer's protocol generators because (1) they are significantly simpler than [BGW88] and combined with our approach yield a fairly simple and straightforward construction of multiparty protocol generators and (2) they can be implemented over any ring and not just a field. We stress that we improve upon Maurer's protocol generators in that we produce protocols of length polynomial, rather than exponential, in the desired threshold of corrupted processors.

The main caveat of our approach is that it is either (1) based on an unproven conjecture (the majority from majorities conjecture - Conjecture 1) or (2) uses a randomized construction or (3) supports a non-optimal threshold of corrupt processors.

As noted above, we only require the following special cases of Maurer's protocol generator:

Theorem 12 ([Mau06])**.** *There exists an explicit three processor local protocol generator in the* Ring-MPC *model that is secure against a single* passive *adversary.*

Using Lemma 9.1 combined with Theorem 12 and our majority formulae constructions (see Section 6) we obtain Theorems 13, 14 and 15.

Theorem 13. *If the majority from majorities conjecture (Conjecture 1) holds then there exists an explicit protocol generator in the* Ring-MPC *model that is secure against a passive adversary that controls any $t < \frac{n}{2}$ of the n processors.*

Given a protocol for n processors that involves t ring operations, the protocol generator outputs a protocol involving $t \cdot \mathrm{poly}(n)$ ring operations.[24]

Proof. The majority from majorities conjecture implies that there exists an algorithm that on input n outputs a logarithmic depth formula F_n composed of Maj_3 gates (i.e., Th_2^3 gates) that computes majority. The running time of the algorithm is polynomial in n.

Given a specification (π, τ) that involves n processors, the protocol generator constructs F_n and then applies Lemma 9.1 while using Maurer's 3-processor local protocol generator of Theorem 12. □

Note that by Theorem 3, the conjecture used in Theorem 13 can be replaced by the assumption that E does not have $2^{\varepsilon n}$- size circuits for some $\varepsilon > 0$, or even more specifically on the existence of sub-exponentially hard one-way functions.

As an additional result, using our construction of formulae that compute majority given sufficient bias (Theorem 2), we obtain the following result which guarantees close to optimal security.

Theorem 14. *There exists an explicit protocol generator in the passive* Ring-MPC *model that is secure against any adversary that controls a $\frac{1}{2} - 2^{-O(\sqrt{\log n})}$ fraction of the n processors.*

[24]Note that the number of communicated ring elements is always upper bounded by the amount of computation.

Given a protocol for n processors that involves t ring operations, the protocol generator outputs a protocol involving $t \cdot \mathrm{poly}(n)$ ring operations.

Proof. By Theorem 2, there exists an algorithm that on input n outputs a logarithmic depth formula F_n composed of Maj_3 such that given any string of relative Hamming weight less than $\frac{1}{2} - 2^{-O(\sqrt{\log n})}$ as input, the formula F_n outputs 0. The running time of A is polynomial in n.

Given a specification (π, τ) that involves n processors, the protocol generator constructs F_n by running A and then applies Lemma 9.1 while using Maurer's 3-processor local protocol generator of Theorem 12. □

Alternatively, the logarithmic depth majority formulae can be obtained using the randomized construction of [Val84] (see also [Gol11b]). This results in the following randomized construction.

Theorem 15. *There exists a randomized construction of a protocol generator in the passive* Ring-MPC *model that is secure against an adversary that controls $t < \frac{n}{2}$ of the n processors.*

Given a protocol for n processors that involves t ring operations, the protocol generator outputs a protocol involving $t \cdot \mathrm{poly}(n)$ ring operations.

Proof. By Theorem 9, there exists a randomized algorithm A that on input n, other than with exponentially small probability, outputs a logarithmic depth formula F_n composed of Maj_3 gates that computes majority. The running time of A is polynomial in n. As the advice string for our protocol generator we use the random coins of A.

Given a specification (π, τ) that involves n processors, the protocol generator generates F_n by running A on the random coins specified by its advice string and then applies Lemma 9.1 while using Maurer's local 3-processor protocol generator of Theorem 12. □

10.2 The Active Model

In this section we show a protocol generator in the Ring-MPC model that is secure against an active adversary. To do so we shall once again use Lemma 9.1, only this time we reduce the n-processor problem to a four processor problem.

In order to solve the four processor case we once again use Maurer's [Mau06] actively secure protocol. Indeed, Maurer gives a simple and elegant protocol that is secure against any adversary that *actively* controls at most one processor.

Unfortunately, Maurer's four processor active protocol requires operations not supported by the blackbox ring model. Specifically, the protocol requires the ability to test for equality and to choose a majority between three values (where a tie is not possible).

To support Maurer's protocol we could potentially add these operations to our model. However, in order to use Lemma 9.1, we would have to be able to simulate such operations done by a virtual processor using other processors (which does not seem easy).

Instead, we take a different approach. We slightly extend our model by allowing *global* variables. That is, variables that exist in the view of all processors (both real and virtual). Every processor may give a global variable a value (for simplicity we assume that the value of each global variable is only set once) and read the values of global variables.

In addition, we allow the parties to do arbitrary computation over global variables. This computation may use oracle access to the underlying ring but only based on the (adversarially chosen) identifiers of ring elements. In particular the number of oracle queries may not depend on the underlying ring. We call this the Active Ring-MPC model.

Note that the Processor Simulation Theorem (Theorem 11) and Lemma 9.1 can be extended to this model. This can be done since an operation over global variables by a virtual processor can be simulated by having each simulating processor do the exact same computation. Note that since the variables are global, each simulating processor has access to these variables.

It is worthwhile to point out that a protocol in the Active Ring-MPC model can be easily transformed to work in a more standard model in which (1) we allow broadcasts and (2) allow additional operations such as testing equality and taking majority. However, it will be useful for us to present our protocols in the Active Ring-MPC and they can later be adapted to other models.

We give a sketch of the steps required to adapt Maurer's four processor protocol to the Active Ring-MPC model:

1. **Handling Equality:** equality is used in Maurer's protocol only in the consistency check of the underlying verifiable secret sharing (VSS) protocol.

   In the four processor VSS protocol, a dealer sharing a secret $s \in R$ sends shares s_1, s_2, s_3, s_4 to processors p_1, p_2, p_3, p_4 such that processor i receives all shares but s_i. Then, for every $i \in \{1, 2, 3, 4\}$, every processor except the i-th processor check that their received value s_i is the same. This is done by three pairwise equality tests.

   Suppose that processors j and k holding the respective shares $s_i^{(j)}$ and $s_i^{(k)}$ (which are supposed to be equal to s_i) want to verify that their share are equal. In Maurer's protocol this is done by simply sending each other the shares and broadcasting a complaint if they differ.

   Instead of testing equality, both the j-th and k-th processor publish the difference $s_i^{(j)} - s_i^{(k)}$ between their two shares as a global variable. We argue that this does not extend the view of the adversary. Indeed, if the adversary controls either processor i, j or k then its view is not extended since it already knows both $s_i^{(j)}$ and $s_i^{(k)}$. If the adversary controls the fourth processor then its view is also not extended since it will always see the constant 0 (recall that since we deal with a four processor protocol there is only one processor controlled by the adversary).

   Suppose that one of these global variables $v_{i,j,k}$ corresponding to a share s_i was set to a non-zero value. Then, as in Maurer's protocol, the dealer publishes s_i as a global value which is used by all parties as the correct value of s_i. Note that the latter operation can be implemented solely using global variables and computation over these global variables.

2. **Handling Majority:** A majority between three value $a, b, c \in R$ (where there is no possibility of a tie) is used in Maurer's VSS reconstruction phase. In this protocol each processor computes the majority of three values that were broadcast by three processors. We replace these broadcasts by publishing global variables and

therefore the majority operation is an operation done over global variables which we allow in our model.

Using this transformation, we can state Maurer's active protocol in terms of a four-processor protocol generator:

Theorem 16 ([Mau06])**.** *There exists an explicit four processor local protocol generator in the* Active Ring-MPC *model that is secure against an active adversary that controls a single processor.*

Using Theorem 16 combined with Lemma 9.1 and our construction of a logarithmic depth threshold formula from Th_2^4 gates (Theorem 5), we obtain the following result in the active model.

Theorem 17. *There exists an explicit protocol generator in the* Active Ring-MPC *model that is secure against an active adversary that controls at most a $\frac{1}{3} - \Omega(\frac{1}{\sqrt{\log n}})$ fraction of the n processors.*

Given a protocol for n processors that involves t ring operations, the protocol generator outputs a protocol involving $t \cdot \mathrm{poly}(n)$ ring operations.

Proof. By Theorem 5, there exists an algorithm that on input n outputs a logarithmic depth formula F_n composed of Th_2^4 gates that outputs 0 for any input of relative Hamming weight $\frac{1}{3} - \Omega(\frac{1}{\sqrt{\log n}})$ or less.

Given a specification (π, τ) that involves n processors, the protocol generator generates F_n and then applies Lemma 9.1 while using Maurer's 4-processor local protocol generator of Theorem 16. □

10.3 MPC over k-Linear Maps

In this section we describe an extension of Maurer's protocol generator which supports blackbox computation over an arbitrary basis of k-linear maps. For simplicity, we restrict the attention here to k-linear maps over Abelian groups (rather than vector spaces or modules, over which they are usually defined). The computational model is defined by a set of finite Abelian groups $G_1, \dots, G_m$ written in additive notation and a basis B of k-linear maps over these groups, where a k-linear map is a function $L : G_{i_1} \times \cdots \times G_{i_k} :\to G_{i_0}$ with the following property. If all of the input variables but the j-th are held constant, then the resulting function $L' : G_{i_j} \to G_{i_0}$ satisfies $L'(g + g') = L'(g) + L'(g')$ for each $g, g' \in G_{i_j}$. Note that k'-linear maps for $k' < k$ (including addition in a single group G_i) are special cases of k-linear maps. A blackbox computation over B can have inputs and variables taken from any of the groups G_i and may combine them using an arbitrary sequence of k-linear maps from B as well as individual group operations.

Our main observation is that by using a simple generalization of Maurer's protocol, $n = k + 1$ processors (resp., $n = k + 2$ processors) suffice for evaluating an arithmetic circuit over B with security against a single passively (resp., actively) corrupted processor. We sketch the approach for the passive case.

As in [Mau06], we represent each group element $g \in G$ by $k + 1$ additive shares $g_1, \dots, g_{k+1}$ such that processor i holds all shares *except* g_i. Now, suppose we are given k

group elements $g^{(1)}, \dots, g^{(k)}$ represented in this way, so that $g^{(i)} = \sum_{j=1}^{k+1} g_j^{(i)}$. To locally compute additive shares of $L(g^{(1)}, \dots, g^{(k)})$, we write

$$L(g^{(1)}, \dots, g^{(k)}) = \sum_{a \in [k+1]^k} L\left(g_1^{(a_1)}, \dots, g_k^{(a_k)}\right).$$

Noting that the value of each of the $(k+1)^k$ terms is known to at least one of the $k+1$ processors (namely, any processor whose index does not occur in the tuple a corresponding to the term), we can assign each term to a processor who can evaluate it. Letting each processor sum the values of the term assigned to it, we get an additive representation of $L(g^{(1)}, \dots, g^{(k)})$. To continue the computation, each share of the output needs to be re-shared as in the protocol of [Mau06].

This approach, combined with Theorem 5, yields the following theorem.

Theorem 18. *For any constant $k \geq 2$, there exists an explicit protocol generator in the model of MPC over k-linear maps that is secure against a passive (resp., active) adversary controlling at most a $\frac{1}{k} - \Omega(\frac{1}{\sqrt{\log n}})$ (resp., $\frac{1}{k+1} - \Omega(\frac{1}{\sqrt{\log n}})$) fraction of the n processors.*

Given a protocol for n processors that involves t group operations, the protocol generator outputs a protocol involving $t \cdot \mathrm{poly}(n)$ group operations.

11 Secure MPC over Groups

In this section we consider a different instantiation of the abstract MPC framework introduced in Section 8, where the computation is done over a finite group while only making a black-box access to the group. In particular, the number of group elements communicated during the protocol does not depend on the computational complexity of the group operation. This model was introduced by Desmedt *et al.* [DPSW07] and further studied in [SYT08, DPS+12b, DPS12a].

Let $(G, *)$ be a fixed finite group written in multiplicative notation. We do not assume that the group is Abelian (and think of it as being non-Abelian). We instantiate the framework with variables taking values in G. We allow the processors to compute the following operations:

- The group operator $*$. That is, given $g_1, g_2 \in G$ a processor can compute the value $g_1 * g_2$. Note that since the group is (usually) non-Abelian, the order of operands is important.
- The operator *invert* which given an operand $g \in G$ returns the inverse g^{-1} of g.
- The operator *rand* that takes no operands and returns a uniformly distributed group element.

We call this model the *Group-MPC model.* For brevity, we will sometimes write $g_1 g_2$ instead of $g_1 * g_2$.

11.1 The Passive Model

In this section, we directly present a simple 3-processor protocol generator (in the Group-MPC model) that has passive security against a single adversary. The protocol that we present is loosely based on a protocol of Feige, Killian and Naor [FKN94] and simplifies a previous protocol from [DPS+12b].

Let $x \in G$. Recall that a 2-out-of-2 secret sharing of x is the following (random) process: select $x_1 \in_R G$ and set $x_2 = x_1^{-1}x$ such that $x = x_1x_2$). We call (x_1, x_2) a sharing of x. Note that since the group may be non-Abelian, x_1 and x_2 play different roles and are called the left and right shares respectively.

Our protocol consists of three processors p_1, p_2, p_3. The protocol generator will maintain the invariant that every variable held in the ideal protocol (i.e., the specification) by τ is secret shared by p_1 and p_2 such that p_1 holds the left share and p_2 holds the right share. In fact, by "computing a sharing z" we mean that p_1 computes z_1 and p_2 computes z_2 such that $z = z_1z_2$ and z_1 is uniformly distributed in G.

Before presenting the 3-processor protocol generator, we present two useful sub-protocols that will be used by the protocol generator. The first protocol is useful for multiplying two sharings. It is described and proved in Claim 11.1. The second protocol transforms a sharing (a, b) into a random sharing of ba. It is described and proved in Claim 11.2.

Claim 11.1 (Share Multiplication Protocol)**.** *Let $\{p_1, p_2, p_3\}$ be a set of three processors in the Group-MPC model such that p_1 gets as input $a_1, a_2 \in G$, p_2 gets as input $b_1, b_2 \in G$ and p_3 has no input. Let $z = a_1b_1a_2b_2$. Then there exists a protocol for computing a sharing of z such that the view of each processor is statistically independent of the input of the other processors.*

Proof. Consider the following protocol:

1. p_1 selects at random $r_0, r_1, r_2, r_3 \in_R G$ and sends r_0, r_1, r_2, r_3 to p_2.
2. p_1 computes $a'_1 = r_0^{-1}a_1r_1$ and $a'_2 = r_2^{-1}a_2r_3$ and sends a'_1 and a'_2 to p_3.
3. p_2 computes $b'_1 = r_1^{-1}b_1r_2$ and $b'_2 = r_3^{-1}b_2$ and sends b'_1 and b'_2 to p_3.
4. p_3 selects $r' \in_R G$ and sends $u_1 = a'_1b'_1a'_2b'_2r'$ to p_1 and $u_2 = r'^{-1}$ to p_2.
5. The share of p_1 is $z_1 = r_0u_1$ and the share of p_2 is $z_2 = u_2$.

Note that (z_1, z_2) is indeed a sharing of z since z_2 is uniformly distributed in G and

$$z_1z_2 = r_0u_1u_2 = r_0a'_1b'_1a'_2b'_2 = (r_0r_0^{-1})a_1(r_1r_1^{-1})b_1(r_2r_2^{-1})a_2(r_3r_3^{-1})b_2 = a_1b_1a_2b_2 = z.$$

The view of p_1 consists only of $a_1, a_2, r_0, r_1, r_2, r_3, z_1$. Since $z_1 = r_0a_1b_1a_2b_2r'$, and since r' is not known to p_1, the view is statistically independent from b_1, b_2.

The view of p_2 consists only of $b_1, b_2, r_0, r_1, r_2, r_3, z_2$ which is statistically independent from a_1, a_2 (since $z_2 = r'^{-1}$).

The view of p_3 consists only of $r_0^{-1}a_1r_1, r_1^{-1}b_1r_2, r_2^{-1}a_2r_3, r_3^{-1}b_2$ which is statistically independent of a_1, b_1, a_2, b_2. □

Claim 11.2 (Share Inversion Protocol)**.** *Let $\{p_1, p_2, p_3\}$ be a set of three processors in the* Group-MPC *model such that p_1 gets as input $a \in G$, p_2 gets as input $b \in G$ and p_3 has no input. Then there exists a protocol for computing a sharing of the inverted product ba such that the view of each processor is statistically independent of the input of the other processors.*

Proof. Consider the following protocol:

1. p_1 samples uniformly at random $r_1 \in_R G$, computes $y_1 = ar_1$ and sends r_1 to p_2 and y_1 to p_3.

2. p_2 samples uniformly at random $r_2 \in_R G$, computes $y_2 = r_2 b$ and sends r_2 to p_1 and y_2 to p_3.

3. p_3 samples uniformly at random $s \in_R G$, computes $w_1 = y_2 s$ and $w_2 = s^{-1} y_1$. It sends w_1 to p_1 and w_2 to p_2.

4. The share of p_1 is $z_1 = r_2^{-1} w_1$ and the share of p_2 is $z_2 = w_2 r_1^{-1}$.

Note that:

$$z_1 z_2 = r_2^{-1} w_1 w_2 r_1^{-1} = r_2^{-1} y_2 s s^{-1} y_1 r_1^{-1} = r_2^{-1} r_2 b a r_1 r_1^{-1} = ba$$

as required.

The view of p_1 consists of a, r_1, r_2, w_1. Since $w_1 = r_2 bs$ and s is unknown to p_1, its view is statistically independent of b.

Similarly, the view of p_2 consists of b, r_1, r_2, w_2. Since $w_2 = sar_1$ and s is unknown to p_2, its view is statistically independent of a.

Finally, the view of p_3 consists of $ar_1, r_2 b$. Since r_1 and r_2 are unknown to p_3, its view is statistically independent of a, b. □

Using Claims 11.1 and 11.2 we are ready to describe our protocol generator and prove its security against a single adversary.

Lemma 11.3. *There exists a 3-processor local protocol generator in the* Group-MPC *model that has passive security against a single adversary.*

Proof. Let (π, τ) be a specification for the set of processors $\{p_1, p_2, p_3\}$. Given (π, τ) as input, the protocol generator outputs a protocol π' by replacing each statement of π that involves τ by a sub-protocol. Statements that do not involve τ are mapped to π' as-is. An invariant that we maintain that in π', is that if τ has access to some variable x in π then p_1 and p_2 have access to a sharing of x in π'.

1. Every statement of the form *transmit*(p_i, τ, z) for $i \in \{1, 2, 3\}$ is mapped to the following sub-protocol. The processor p_i selects at random $z_1 \in_R G$ and sets $z_2 = z_1^{-1} z$ (such that $z = z_1 z_2$). It sends z_1 to p_1 and z_2 to p_2. Note that this process maintains our invariant that p_1 and p_2 hold a sharing of z.

2. Every statement of the form *transmit*(τ, p_i, z) for $i \in \{1, 2, 3\}$ is mapped to the following sub-protocol. Since z is in the view of τ, by our invariant, p_1 and p_2 have a sharing (z_1, z_2) of z. Processor p_1 sends z_1 to p_i and processor p_2 sends z_2 also to p_i. Processor p_i then computes $z = z_1 z_2$.

3. Every statement of the form $comp(\tau, *, z_1, z_2, z)$ is mapped to the following sub-protocol. Since z_1 is known to τ, by our invariant, p_1 and p_2 have a sharing (a_1, b_1) of z_1 and a sharing (a_2, b_2) of z_2. To compute a sharing of $z = z_1 * z_2 = a_1 b_1 a_2 b_2$, p_1 and p_2 simply run the share multiplication protocol (Claim 11.1).

4. Every statement of the form $comp(\tau, inverse, z, z')$ is mapped to the following sub-protocol. Since z is known to τ, by our invariant, p_1 and p_2 have a sharing (z_1, z_2) of z. They both invert their shares and run the inversion protocol (Claim 11.2) on the inverted shares. At the end of the protocol p_1 has u_1 and p_2 has u_2 such that $u_1 u_2 = z_2^{-1} z_1^{-1} = (z_1 z_2)^{-1} = z^{-1}$.

5. Every statement of the form $comp(\tau, random, z)$ is mapped to the following sub-protocol. Processor p_1 samples a random element r_1 and p_2 samples a random element r_2. They set $z = (r_1, r_2)$. That is (r_1, r_2) is a sharing of z.

We proceed to show that π' securely computes the specification (π, τ) with respect to an adversary that sees the view of a single processor.

Fix an adversary A' that sees the view of a single processor $p^* \in \{p_1, p_2, p_3\}$ in π' and fix inputs to the three processors. We consider the statement index function that maps each statement in π to the corresponding sub-protocol (described above) in π'. We will show an adversary A such that the joint distribution of its view together with the output of the uncorrupted processors in π is identical to the view of A' together with the output non-corrupted processors in π'.

We proceed to describe what the adversary A does for the i-th statement of π and argue why its view together with the output of the uncorrupted processors in π remains identically distributed to that of A' together with the uncorrupted processors in π'. For every $i \in \{1, \ldots, |\pi| + 1\}$, the adversary A performs the following steps for the i-th statement of π:

1. If the statement (1) does not involve τ or (2) it is of the form $transmit(p_i, \tau, z)$ or (3) is of form $transmit(\tau, p_i, z)$ for $p_i \neq p^*$ then A runs the corresponding sub-protocol in π' and performs the same steps that A' would perform. Since the views of neither A' is not extended and that output of all non-corrupt processors is either unchanged or changed similarly (in case of an *output* statement), the joint distribution of the view of A and the output of the uncorrupted processors in π' remains identical to that of A' and the uncorrupted processors in π.

2. If the statement is of the form $comp(\tau, *, z_1, z_2, z)$ and $p^* \in \{p_1, p_2\}$ then (by Claim 11.1, the view of A' is extended by a random share of z. Thus, A extends its view by simply selecting a uniformly distributed group element. If $p^* = p_3$ then it simply performs the same steps that A' would perform.

3. If the statement is of the form $comp(\tau, inverse, z_1, z)$ and $p^* \in \{p_1, p_2\}$ then the view of A' is extended by an independent random sharing of z^{-1} (by Claim11.2). Thus, A extends its own view by a uniformly distributed group element. If $p^* = p_3$ then it simply performs the same steps that A' would perform.

4. If the statement is of the form $comp(\tau, rand, z)$ then A simulates the corresponding sub-protocol in π'. The view of A' is (possibly extended) by a single share of the random element z. The adversary A simulates this by choosing a random group element in G.

5. If the statement is of the form $transmit(\tau, p^*, z)$ then A simulates the corresponding sub-protocol that recovers z from shares held by p_1 and p_2. It then performs the same steps as A' performs with respect to the above simulation. The view of both A and A' are extended by z.

Thus, after every statement, the joint distribution of the view of A together with the output of all uncorrupted processors in π is identically distributed to that of A' together with the output of the uncorrupted processors in π'. □

Using Lemma 11.3 together with Lemma 9.1 we obtain the following two results:

Theorem 19. *There exists a protocol generator in the passive Group-MPC model that is secure against an adversary that controls a $\frac{1}{2} - 2^{-O(\sqrt{\log n})}$ fraction of the n processors.*

Given a protocol for n processors that involves t group operations, the protocol generator outputs a protocol involving $t \cdot \mathrm{poly}(n)$ group operations.

Proof. By Theorem 2, there exists an algorithm that on input n outputs an $O(\log n)$ depth formula F_n composed of Maj_3 gates such that for every input of normalized weight less than $\frac{1}{2} - 2^{-O(\sqrt{\log n})}$, the formula F_n outputs 0. The theorem follows by an application of Lemma 9.1 based on the 3-processor local protocol generator of Lemma 11.3. □

Theorem 20. *If the majority from majorities conjecture (Conjecture 1) holds then there exists a protocol generator in the Group-MPC model that has passive security against an adversary that controls at most $t < \frac{n}{2}$ processors.*

Given a protocol for n processors that involves t group operations, the protocol generator outputs a protocol involving $t \cdot \mathrm{poly}(n)$ group operations.

Proof. The majority from majorities conjecture implies the existence of an algorithm that on input n outputs a logarithmic depth formula that uses only Maj_3 gates and computes majority. The theorem follows from an application of Lemma 9.1 based on the 3-processor local protocol generator of Lemma 11.3. □

Theorem 21. *There exists a randomized construction of a protocol generator in the passive Group-MPC model that is secure against an adversary that controls $t < \frac{n}{2}$ processors.*

Given a protocol for n processors that involves t group operations, the protocol generator outputs a protocol involving $t \cdot \mathrm{poly}(n)$ group operations.

Proof. By Theorem 9, there exists a randomized polynomial-time algorithm that on input n, other than with exponentially vanishing probability, outputs an $O(\log n)$ depth formula composed of Maj_3 gates that computes majority. The theorem follows by an application of Lemma 9.1 based on the 3-processor local protocol generator of Lemma 11.3. □

Theorem 22. *There exists a protocol generator in the passive Group-MPC model that is secure against an adversary that controls $t < \frac{n}{2}$ processors.*

Given a protocol for n processors that involves t group operations, the protocol generator outputs a protocol involving $t \cdot n^{O(\log n)}$ group operations.

Proof. By Lemma 6.1, there exists a (deterministic) polynomial-time algorithm that on input n, runs in time $n^{O(\log n)}$, and outputs an $O(\log^2 n)$ depth formula (of size $n^{O(\log n)}$) composed of Maj_3 gates that computes majority. The theorem follows by an application of Lemma 9.1 based on the 3-processor local protocol generator of Lemma 11.3. $\square$

11.2 The Active Model

In a recent work, Desmedt *et al.* [DPS12a], gave a protocol (or in our terminology, a protocol generator) in the Group-MPC model which is *actively* secure against any Q_3 adversary. However, the complexity of their protocol generator is quadratic in the number of maximal sets in the adversary structure, which in the case of thresholds, is exponential in the number of processors. In this section we address an open problem stated by [DPS12a] by showing a protocol generator in the Group-MPC that has active security against threshold adversaries where the complexity of the protocol is polynomial in the number of processors. Our protocol is *actively* secure against any adversary that controls at most a $\frac{1}{3} - \Omega(\frac{1}{\sqrt{\log n}})$ fraction of the n processors (in contrast to the optimal threshold of $\frac{1}{3} - \Omega(\frac{1}{n})$).

We basically use the same processor simulation approach formalized in Section 9 and used in previous sections. Recall that in order to use this approach we must rely on a protocol-generator for a constant number of processors. For this we simply use an instantiation of the protocol of Desmedt *et al.* [DPS12a] for four processors that is secure against an adversary that *actively* control one processor.

Unfortunately, as in the case of Maurer's four processor active protocol in the Ring-MPC model (see Section 10.2), the [DPS+12b] protocol uses operations that are not supported by our model. Specifically equality testing and computing majority.

As in Section 10.2, we extend the Group-MPC model by allowing *global* variables and allowing arbitrary computation over global variables. This computation may use oracle access to the underlying group but only based on the (adversarially chosen) identifiers of group elements. In particular, the number of oracle queries may not depend on the underlying ring. We call this the Active Group-MPC model.

Note that a protocol in the Active Group-MPC model can be easily transformed to work in a more standard model in which (1) we allow broadcasts and (2) allow additional operations such as testing equality and taking majority vote.

Below, we give a sketch of the steps required to adapt the four processor [DPS+12b] protocol to the Active Group-MPC model:

1. **Equality type 1:** The first type of equality test that is used in the [DPS+12b] protocol in the consistency check of the underlying VSS and consistency checks in the NodeMult protocol. The problem and its solution are almost identical to that encountered in Section 10.2.

   In this type of equality, a dealer send secret shares of some value and the receiving parties check the pairwise consistency of their shares. Specifically, suppose that processor j and k are sent respective shares $s_i^{(j)}$ and $s_i^{(k)}$ (which are supposed to be equal to a share s_i) and want to verify that their share are equal. In Maurer's protocol this is done by simply sending each other the shares and broadcasting a complaint if they differ.

Instead of testing equality, both the j-th and k-th processor publish the ratio $s_i^{(j)} * (s_i^{(k)})^{-1}$ between their two shares as a global variable. We argue that this does not extend the view of the adversary. Indeed, if the adversary controls either processor i, j or k then its view is not extended since it already knows both $s_i^{(j)}$ and $s_i^{(k)}$. If the adversary controls the fourth processor then its view is also not extended since it will always see the neutral element of G (recall that since we deal with a four processor protocol there is only one processor controlled by the adversary).

Suppose that one of these global variables $v_{i,j,k}$ corresponding to a share s_i was set to a value not-equal to the neutral element. Then, as in the [DPS12a] protocol, the dealer publishes s_i as a global value which is used by all parties as the correct value of s_i. Note that the latter operation can be implemented solely using global variables and computation over these global variables.

2. **Equality type 2:** The second type of equality operation occurs in the second step of the NodeMult protocol. However, in this step after a processor tests whether $a = b$, if $a \neq b$ it just broadcasts the value $a * b^{-1}$. Thus, instead of testing $a = b$ we just set $a * b^{-1}$ as a global variable and use it as specified.

3. **Equality type 3:** The third type of equality operations occurs is on global variables. Since this computation is done over global variables we do not need to change it (recall that we allow arbitrary computation over global variables).

4. **Majority:** A majority between three values $a, b, c \in R$ (where there is no possibility of a tie) is used in the [DPS12a] protocol only when a processor computes the majority of three values that were broadcast by three processors. We replace these broadcasts by publishing global variables and therefore the majority operation is an operation done over global variables which we allow in our model.

Thus, we have the following theorem:

Theorem 23 ([DPS12a])**.** *There exists an explicit four processor local protocol generator in the Active Group-MPC model that is secure against an active adversary that controls a single processor.*

Using Theorem 16 combined with Lemma 9.1 and our construction of a logarithmic depth threshold formula from Th_2^4 gates (Theorem 5), we obtain the following result in the active model.

Theorem 24. *There exists an explicit protocol generator in the Active Group-MPC model that is secure against an active adversary that controls at most $\frac{1}{3} - \Omega(\frac{1}{\sqrt{\log n}})$ of the n processors.*

Given a protocol for n processors that involves t group operations, the protocol generator outputs a protocol involving $t \cdot \mathrm{poly}(n)$ group operations.

Proof. By Theorem 5, there exists an algorithm that on input n outputs a logarithmic depth formula F_n composed of Th_2^4 gates that outputs 0 for any input of normalized Hamming weight $\frac{1}{3} - \Omega(\frac{1}{\sqrt{\log n}})$ or less.

Given a specification (π, τ) that involves n processors, the protocol generator generates F_n and then applies Lemma 9.1 while using the [DPS12a] four processor local protocol generator of Theorem 23. □

11.3 Two-Party Protocols

In this section we establish the first feasibility results for secure *two-party* computation over black-box groups. The result we get for the active corruption model illustrates the usefulness of the "threshold from threshold" technique even in the context of two-party cryptography.

Instead of basing our protocols on concrete cryptographic assumptions, we follow the convention of allowing the parties access to an ideal oblivious transfer oracle. Recall that an oblivious transfer (OT) [Rab81, EGL85] oracle computes a two-party function that allows a receiver to obtain one out of two strings held by a sender, without revealing to the sender which of the two strings it chose. We refer to secure computation in the presence of an ideal OT oracle as secure computation in the OT-Hybrid model. The advantage of working in the OT-Hybrid model is that it enables unconditional security. Using composition theorems for secure computation, the ideal OT oracle can be replaced by (a black-box access to) any secure OT implementation.

Our two-party protocols are *statistically* secure in the OT-Hybrid model. That is, the parties are given a security parameter 1^k and we require that for every adversary, there exists an adversary in the ideal world whose view is $\exp(-k)$-close to that of the real world adversary. The running time of the two parties may depend polynomially on k and polylogarithmically on (an upper bound on) the group size. Since the latter may be inferred from the length of the identifiers of group elements in the standard generic group model, this convention does not violate the "black-box" aspect of the model (and in particular does not allow the protocol to learn the group structure).

In Section 11.3.1 we show a secure a two party passive protocol and in Section 11.3.2 we show, using the compiler of Ishai, Prabhakaran and Sahai [IPS08], how to transform the passive protocol together with the unconditionally secure active protocol of Section 11.2 into a two party actively secure protocol.

11.3.1 A Two-Party Passively Secure Protocol

We denote the two parties by sender and receiver. The secure evaluation of a general circuit over a group reduces to securely computing an iterated group product of the form $c = a_1 b_1 a_2 b_2 \cdots a_m b_m$ where a_i are the sender's inputs, b_i are the receiver's inputs, and the receiver gets the output c. (Indeed, this suffices for generating random shares of the product xy given shares of x and shares of y.)

We will show a protocol for computing such an iterated group product in the OT-Hybrid model where the view of each party can be simulated, up to $2^{-\Omega(k)}$ statistical distance, given its input and output.

Before proceeding to the protocol we state and prove a lemma that will be useful in the analysis of our protocol. The lemma generalizes a lemma of Impagliazzo and Naor [IN96] on the subset sum problem. Recall that the *statistical distance* between two distributions $\mathcal{D}_1, \mathcal{D}_2$ with support S is defined as $\mathsf{SD}(\mathcal{D}_1, \mathcal{D}_2) \triangleq \max_{A \subseteq S} |\mathsf{Pr}[\mathcal{D}_1 \in S] - \mathsf{Pr}[\mathcal{D}_2 \in S]|$.

Lemma 11.4. *Let G be a finite group, let $k > 0$ be a security parameter and let $s = 4k + \log(|G|)$. Then, except for a 2^{-k} fraction of $r_1, \ldots, r_s \in G$ it holds that:*

$$\mathsf{SD}(r_1^{w_1} \cdots r_s^{w_s}, g) < 2^{-k},$$

where $w_1, \ldots, w_s$ are uniformly distributed bits and g is uniformly distributed in G.

Proof. Consider the universal hash function family $\{h_{r_1,\ldots,r_s} : \{0,1\}^s \to G\}_{r_1,\ldots,r_s \in G}$ defined as $h_{r_1,\ldots,r_s}(w) = \prod_{i=1}^{s} r_i^{w_i}$. Then:

$$\mathop{\mathbf{E}}_{r_1,\ldots,r_s \in G}\left[\mathsf{SD}_{\substack{w\in\{0,1\}^s\\ g\in G}}(h_{r_1,\ldots,r_s}(w), g)\right] = \mathsf{SD}_{\substack{g,r_1,\ldots,r_s\in G\\ w\in\{0,1\}^s}}\Big((r_1,\ldots,r_s,h_{r_1,\ldots,r_s}(w)),(r_1,\ldots,r_s,g)\Big)$$

which, by the Leftover Hash Lemma[25] [HILL99], is at most $\sqrt{\frac{|G|}{2^s}} = 2^{-2k}$. The lemma follows by an application of Markov's inequality. □

Lemma 11.4 is used to get a statistical secret sharing of group elements in the following way. The receiver publishes $s = 4k + \log|G|$ public random group elements $r_1, \ldots, r_s$. Now, for any $w \in \{0,1\}^s$ we can represent a group element g using (w, g') where $g' = g * \prod r_i^{w_i}$. By the lemma, if we pick w at random then, except with 2^{-k} probability over the choice of r_i, the element g' is 2^{-k}-close to uniform even when conditioned on the r_i. This means that (w, g') form a statistical 2-out-of-2 secret sharing of g given public randomness $(r_1, \ldots, r_s)$.

Using the above method for secret sharing group elements, the iterated group product protocol proceeds as follows:

1. The receiver picks $r_1, \ldots, r_s \in G$ uniformly at random and sends $r_1, \ldots, r_s$ to the sender.

2. Using these r_i, the receiver splits each of his inputs b_i into (w_i, b'_i) and send b'_i to the sender. By Lemma 11.4, the view of the sender can be simulated up to $2^{-\Omega(k)}$ statistical distance without knowing b_i.

3. The sender can now write c as an iterated group product in which some slots include his own inputs a_i, some slots include the values b'_i, and the rest include one of two options: either 1 or an element r_j^{-1}. The choice between the two options is determined by the bit w_i known to the receiver.

4. The sender randomizes this product using Kilian's group product randomization technique [Kil88]. That is, we first write the product as $g_1(c_1) * \cdots * g_n(c_n)$ where each g_i is some function from $\{0,1\}$ to G known to the sender and c_i is a choice bit known to the receiver. The sender then picks random group elements $q_1, \ldots, q_{n-1}$ and lets $g'_1(c_1) = g_1(c_1)q_1, g'_2(c_2) = q_1^{-1}g_2(c_2)q_2, \ldots, g'_n(c_n) = q_{n-1}^{-1}g_n(c_n)$.

5. The sender and receiver invoke the OT oracle n times, delivering to the receiver the values $g'_i(c_i)$ where c_i are the receiver's choice bits. The n received group elements contain no more information than $z = \prod g_i(c_i)$.

The above protocol implies the following theorem.

Theorem 25. *Let f be a two-party functionality defined by a circuit of size s over a group G. Then f can be realized in the* OT*-Hybrid model by a protocol which has statistical security against one passively corrupted party. The protocol requires a total of $O((k + \log|G|)s)$ group operations for achieving 2^{-k} simulation error.*

[25] A simplified version of the Leftover Hash Lemma states that if h is selected at random from a universal hash function family from $\mathcal{X}$ to $\mathcal{Y}$ then the distribution $(h, h(\mathcal{X}))$ and $(h, \mathcal{Y})$ are at most $\sqrt{\frac{|\mathcal{Y}|}{|\mathcal{X}|}}$-close (see, e.g., [Gol08, Appendix D]).

11.3.2 A Two Party Actively Secure Protocol

To get active security in the two-party model, we use the general compiler of [IPS08]. This compiler yields a 2-party protocol for a function f with statistical security against active corruption in the OT-hybrid model by making a *blackbox* use of the following two ingredients:

1. an "outer protocol" for f which employs k auxiliary parties (servers) in addition to the two parties (clients) holding the inputs; this protocol should be perfectly or statistically secure against active corruption provided that only some constant fraction the servers can be corrupted; and

2. an "inner protocol" for implementing a reactive two-party functionality ("inner functionality") corresponding to the local computation of each server, in which the server's state is secret-shared between the two clients. In contrast to the outer protocol, this protocol only needs to be secure against *passive* corruption. The inner protocol can be implemented in the OT-Hybrid model.

We instantiate the outer protocol with the efficient protocol from Theorem 24 and the inner protocol with the two-party protocol from Theorem 25. The details of this combination are analogous to those used for secure two-party computation over black-box rings in [IPS09]. The crucial observation is that when the parties in the outer protocol only make a blackbox use of a group G, then the functionality realized by the inner protocol is also cast in the blackbox model and so parties in the inner protocol can make a blackbox use of G. We refer the reader to [IPS09] for more details.

The result for the active two-party model is summarized by the following theorem.

Theorem 26. *Let f be a two-party functionality defined by a circuit of size s over a group G. Then f can be realized in the OT-Hybrid model by a protocol which has statistical security against one actively corrupted party. The protocol requires a total of* $\mathrm{poly}(k) \cdot \log |G| \cdot s$ *group operations for achieving* 2^{-k} *simulation error.*

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach.* Cambridge University Press, 2009.

[AKS83] Miklós Ajtai, János Komlós, and Endre Szemerédi. An o(n log n) sorting network. In *STOC*, pages 1–9, 1983.

[AR63] Sheldon Akers and Theodore Robbins. Logical design with three-input majority gates. *Computer Design*, 45(3):12–27, 1963.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.

[BIW10] Omer Barkol, Yuval Ishai, and Enav Weinreb. On locally decodable codes, self-correctable codes, and t-private PIR. *Algorithmica*, 58(4):831–859, 2010.

[BM84] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.*, 13(4):850–864, 1984.

[BM92] Peter Bro Miltersen. Lecutre notes. Available from author, 1992.

[Bra87] Gabriel Bracha. An O(log n) expected rounds randomized byzantine generals protocol. *J. ACM*, 34(4):910–920, 1987.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *STOC*, pages 11–19, 1988.

[CDN12] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing - An Information Theoretic Appoach.* 2012. Book draft, available at `http://www.daimi.au.dk/~ivan/MPCbook.pdf`.

[CFF$^+$05] Jeffrey Considine, Matthias Fitzi, Matthew K. Franklin, Leonid A. Levin, Ueli M. Maurer, and David Metcalf. Byzantine agreement given partial broadcast. *J. Cryptology*, 18(3):191–217, 2005.

[CFIK03] Ronald Cramer, Serge Fehr, Yuval Ishai, and Eyal Kushilevitz. Efficient multi-party computation over rings. In *EUROCRYPT*, pages 596–613, 2003.

[Cha89] David Chaum. The spymasters double-agent problem: Multiparty computations secure unconditionally from minorities and cryptographically from majorities. In *CRYPTO*, pages 591–602, 1989.

[DIK$^+$08] Ivan Damgård, Yuval Ishai, Mikkel Krøigaard, Jesper Buus Nielsen, and Adam Smith. Scalable multiparty computation with nearly optimal work and resilience. In *CRYPTO*, pages 241–261, 2008.

[Dol82] Danny Dolev. The byzantine generals strike again. *J. Algorithms*, 3(1):14–30, 1982.

[DPS12a] Yvo Desmedt, Josef Pieprzyk, and Ron Steinfeld. Active security in multiparty computation over black-box groups. In *SCN*, pages 503–521, 2012.

[DPS$^+$12b] Yvo Desmedt, Josef Pieprzyk, Ron Steinfeld, Xiaoming Sun, Christophe Tartary, Huaxiong Wang, and Andrew Chi-Chih Yao. Graph coloring applied to secure computation in non-abelian groups. *J. Cryptology*, 25(4):557–600, 2012.

[DPSW07] Yvo Desmedt, Josef Pieprzyk, Ron Steinfeld, and Huaxiong Wang. On secure multi-party computation in black-box groups. In *CRYPTO*, pages 591–612, 2007.

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *CACM: Communications of the ACM*, 28, 1985.

[FKN94] Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In *STOC*, pages 554–563, 1994.

[FM98] Matthias Fitzi and Ueli M. Maurer. Efficient byzantine agreement secure against general adversaries. In *DISC*, pages 134–148, 1998.

[FM00] Matthias Fitzi and Ueli M. Maurer. From partial consistency to global broadcast. In *STOC*, pages 494–503, 2000.

[GM96] Arvind Gupta and Sanjeev Mahajan. Using amplification to compute majority with small majority gates. *Computational Complexity*, 6(1):46–63, 1996.

[GM98] Juan A. Garay and Yoram Moses. Fully polynomial byzantine agreement for $n > 3t$ processors in $t + 1$ rounds. *SIAM J. Comput.*, 27(1):247–290, 1998.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, pages 218–229. ACM, 1987.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications.* Cambridge University Press, New York, NY, USA, 2004.

[Gol08] Oded Goldreich. *Computational complexity - a conceptual perspective.* Cambridge University Press, 2008.

[Gol11a] Oded Goldreich. A sample of samplers: A computational perspective on sampling. In Oded Goldreich, editor, *Studies in Complexity and Cryptography*, volume 6650 of *Lecture Notes in Computer Science*, pages 302–332. Springer, 2011.

[Gol11b] Oded Goldreich. Valiant's polynomial-size monotone formula for majority. http://www.wisdom.weizmann.ac.il/~oded/PDF/mono-maj.pdf, 2011.

[HIKN08] Danny Harnik, Yuval Ishai, Eyal Kushilevitz, and Jesper Buus Nielsen. OT-combiners via secure computation. In *TCC*, pages 393–411, 2008.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28:12–24, 1999.

[HM00] Martin Hirt and Ueli M. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *J. Cryptology*, 13(1):31–60, 2000.

[HMP06] Shlomo Hoory, Avner Magen, and Toniann Pitassi. Monotone circuits for the majority function. In *APPROX-RANDOM*, pages 410–425, 2006.

[IKOS09] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge proofs from secure multiparty computation. *SIAM J. Comput.*, 39(3):1121–1152, 2009.

[IN96] Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. *J. Cryptology*, 9(4):199–216, 1996.

[INW94] Russell Impagliazzo, Noam Nisan, and Avi Wigderson. Pseudorandomness for network algorithms. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing, STOC 1994*, pages 356–364. ACM, 1994.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO*, pages 572–591, 2008. Preliminary full version in `http://www.cs.illinois.edu/∼mmp/research.html`.

[IPS09] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In *TCC*, pages 294–314, 2009.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988.

[KLR10] Eyal Kushilevitz, Yehuda Lindell, and Tal Rabin. Information-theoretically secure protocols and security under composition. *SIAM J. Comput.*, 39(5):2090–2112, 2010.

[KPS85] Richard M. Karp, N. Pippinger, and Nicholas Sipser. A time-randomness tradeoff. In *AMS Conference on Probabilistic Computational Complexity*, 1985.

[LOP11] Yehuda Lindell, Eli Oxman, and Benny Pinkas. The IPS compiler: Optimizations, variants and concrete efficiency. In *CRYPTO*, pages 259–276, 2011.

[LRM10] Christoph Lucas, Dominik Raub, and Ueli M. Maurer. Hybrid-secure mpc: trading information-theoretic robustness for computational privacy. In *PODC*, pages 219–228, 2010.

[Mau06] Ueli M. Maurer. Secure multi-party computation made simple. *Discrete Applied Mathematics*, 154(2):370–381, 2006.

[MV13] Eric Miles and Emanuele Viola. Shielding circuits with groups. *IACR Cryptology ePrint Archive*, 2013:1, 2013. To appear in STOC 2013.

[Nis92] Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 1994.

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

[Rab81] Michael O. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981.

[RBO89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In David S. Johnson, editor, *STOC*, pages 73–85. ACM, 1989.

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

[SYT08] Xiaoming Sun, Andrew Chi-Chih Yao, and Christophe Tartary. Graph design for secure multiparty computation over non-abelian groups. In *ASIACRYPT*, pages 37–53, 2008.

[Uma03] Christopher Umans. Pseudo-random generators for all hardnesses. *J. Comput. Syst. Sci.*, 67(2):419–440, 2003.

[Vad11] Salil P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 2011.

[Val84] Leslie G. Valiant. Short monotone formulae for the majority function. *J. Algorithms*, 5(3):363–366, 1984.

[Yao82a] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164, 1982.

[Yao82b] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *FOCS*, pages 80–91, 1982.

[Zwi96] Uri Zwick. Lecture notes. http://www.cs.tau.ac.il/~zwick/circ-comp-new/six.ps, 1996.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de