Degree 2 is Complete for the Round-Complexity of Malicious MPC [⋆]

Benny Applebaum[1], Zvika Brakerski[2], and Rotem Tsabary[2]

[1] Tel-Aviv University[⋆⋆]
[2] Weizmann Institute of Science[⋆⋆⋆]

Abstract. We show, via a non-interactive reduction, that the existence of a secure multi-party computation (MPC) protocol for degree-2 functions implies the existence of a protocol with the *same round complexity* for general functions. Thus showing that when considering the round complexity of MPC, it is sufficient to consider very simple functions.
Our completeness theorem applies in various settings: information theoretic and computational, fully malicious and malicious with various types of aborts. In fact, we give a master theorem from which all individual settings follow as direct corollaries. Our basic transformation does not require any additional assumptions and incurs communication and computation blow-up which is polynomial in the number of players and in $S, 2^D$, where S, D are the circuit size and depth of the function to be computed. Using one-way functions as an additional assumption, the exponential dependence on the depth can be removed.
As a consequence, we are able to push the envelope on the state of the art in various settings of MPC, including the following cases.

- 3-round perfectly-secure protocol (with guaranteed output delivery) against an active adversary that corrupts less than 1/4 of the parties.
- 2-round statistically-secure protocol that achieves security with "selective abort" against an active adversary that corrupts less than half of the parties.
- Assuming one-way functions, 2-round computationally-secure protocol that achieves security with (standard) abort against an active adversary that corrupts less than half of the parties. This gives a new and conceptually simpler proof to the recent result of Ananth et al. (Crypto 2018).

Technically, our non-interactive reduction draws from the encoding method of Applebaum, Brakerski and Tsabary (TCC 2018). We extend these methods to ones that can be meaningfully analyzed even in the presence of malicious adversaries.

[⋆] Full version available at https://eprint.iacr.org/2019/200.

[⋆⋆] Email: bennyap@post.tau.ac.il. Supported by the European Union's Horizon 2020 Programme (ERC-StG-2014-2020) under grant agreement no. 639813 ERC-CLC, and the Check Point Institute for Information Security.

[⋆⋆⋆] Email: {zvika.brakerski,rotem.tsabary}@weizmann.ac.il. Supported by the Israel Science Foundation (Grant No. 468/14), Binational Science Foundation (Grants No. 2016726, 2014276), and by the European Union Horizon 2020 Research and Innovation Program via ERC Project REACT (Grant 756482) and via Project PROMETHEUS (Grant 780701).

1 Introduction

A secure multi-party computation (MPC) allows a collection of n parties to jointly compute a function f of their joint inputs without leaking additional information other than the output. The focus of this work is on the so-called "malicious" setting, where security should be guaranteed even if an adversary, that controls up to t parties, *actively* deviates from the protocol instructions. Security is usually formalized using the ideal vs. real paradigm, essentially translating an adversarial behavior in the protocol (the real model) into an indistinguishable behavior in a model where f is computed by a trusted party (the ideal model). Various flavors of this notion have been considered in the literature, but since our results apply in multiple settings we wish to keep the discussion general at this point and not commit to a specific ideal model (however, we do focus on the case where there are private channels between the parties).

A significant resource to optimize when designing an MPC protocol is the *round complexity*, the number of communication rounds that are required in order to complete the protocol (as usual, we assume simultaneous message transmission in each round). Efforts to minimize round complexity started as soon as MPC was introduced [24] and are receiving a lot of attention recently as well (e.g. [1,6,9,10] and many others). A prominent approach to reducing round complexity is tied to reducing the *algebraic degree* of the function to be computed.[3] This can be traced back to the work of Beaver, Micali and Rogaway [4,22] and to the *randomizing polynomials* approach of Ishai and Kushilevitz [15,16]. The latter work is based on the following paradigm: Reduce the task of securely computing the function f to the task of securely computing a different function h, such that h has low algebraic degree, and such that the output of h can be decoded to produce the appropriate output for f. Once such reduction exists, with adequate security guarantees (as we elaborate below), one can focus on providing a secure MPC protocol for h, a task that usually gets easier as the degree of h drops.

In this context, it is most desirable to present a *non-interactive* reduction. Such a reduction yields a function h together with a set of (possibly randomized) local preprocessing function ℓ_i, and a method to decode the value $h(\ell_1, \ldots, \ell_n)$ to recover the output of f. In terms of security, one has to show that the protocol where each party computes ℓ_i locally, sends the value to a trusted realization of h ("an h oracle"), and then performs the decoding of the output locally, is a secure MPC protocol for computing f, respective to a security model specified in the proof.

The resemblance of the h-oracle-aided protocol to the "ideal model" described above allows to compose the reduction with a secure implementation of h, resulting in a secure realization of f. Since the reduction is non-interactive, the round complexity of the resulting protocol is the same as the round-complexity of (the low degree function) h.

[3] In this work we consider the algebraic degree over the binary field. This is the common setting, but one could consider working over other fields as well.

Given this paradigm, it is natural to ask how low can the degree of h be while still allowing a reduction from any arbitrary f. It is not hard to verify that a linear (i.e. degree 1) h cannot be used to compute general functions. However, the randomizing polynomials approach seems to only imply h of degree 3.[4] Recently, Applebaum, Brakerski and Tsabary [3], showed how to reduce any function to degree-2, but their reduction was only secure in a semi-honest setting, where the adversary is required to follow the protocol (i.e. to compute the functions ℓ_i correctly using a properly sampled random tape). Nevertheless, the [3] reduction allowed them to improve the round complexity of semi-honest MPC with perfect security and with honest majority to the optimum of 2 rounds. The question whether there is non-interactive reduction to a quadratic function in the malicious setting, and the implications of such reduction on the round complexity of malicious MPC, remained open and is addressed in this work.

1.1 Our Results

We show that in various settings, a non-interactive reduction to a degree-2 function is possible. This means that it is sufficient to design protocols for degree-2 functions in order to optimize round complexity. We then design round-optimal protocols by constructing round-efficient protocols for degree-2 functions in some of these settings. Our results are all derived using a single "master theorem", which we believe can serve as basis for deriving additional results in other settings as well. We elaborate on these contributions below.

A Master Non-Interactive Reduction (Section 4). The technical heart of our result is a generic non-interactive "master reduction" from any function f to a degree-2 function h. Methodologically, we show how to convert any protocol Π for computing f (irrespective of round complexity) into a non-interactive h-oracle-aided protocol $\hat{\Pi}$ (we denote this by $\hat{\Pi}^h$), while preserving the security properties of Π. Specifically, we show that any adversarial strategy in the h-oracle-aided protocol can be (perfectly) simulated by an adversary against the protocol Π. In terms of the ideal/real paradigm, we show that for any $\hat{\Pi}^h$ adversary there exists a Π adversary with an identical real model view. We believe that this could be an instrumental tool in constructing and analyzing MPC protocols, since it allows to translate arbitrary protocols to ones that make a single oracle call to a fairly simple function.

We note that the conversion between Π and $\hat{\Pi}^h$ incurs a communication and computation overhead that is polynomial in (roughly) the total computational complexity of Π (i.e. the sum of computational complexities of all parties participating in Π throughout the execution of the protocol) and exponential in the depth of Π (roughly the longest computational path between an input and an output in the protocol). Therefore, if communication and computational

[4] It is known that general functions cannot be represented by degree-2 *perfectly-private* randomizing polynomials [15]. The existence of statistically-private degree-2 randomizing polynomials has been open for nearly two decades.

complexity are of importance, we must be careful to only apply our theorem on fairly "shallow" protocols Π.

Our master theorem generalizes the "multi-party randomized encoding" (MPRE) approach of [3], both in terms of theorem statement and in terms of techniques. The notion of h-oracle-aided protocol converges in the semi-honest setting to MPRE, but allows to handle malicious adversarial behavior whereas MPRE is by default a "passive" notion. Our master theorem, while building on the techniques of [3], also implies their MPRE result as a special case since a semi-honest $\hat{\Pi}^h$ adversary translates to a semi-honest Π adversary.

A Completeness Theorem (Section 5). To illustrate the power of our master theorem, we show a non-interactive reduction from the task of computing an arbitrary function f to the task of computing degree-2 functions, in the context of full security (guaranteed output delivery):

- A perfectly secure reduction, assuming more than 2/3 of the parties are honest.
- A statistically secure reduction, assuming more than 1/2 of the parties are honest.
- A computationally secure reduction, assuming more than 1/2 of the parties are honest, and assuming the existence of one-way functions (that are used in a black-box manner).

All of those reductions incur a communication and computation overhead. In all reductions this overhead is polynomial in the number of parties and in the size of the circuit computing f, and in the first two reductions (i.e. without making computational assumptions) it is also exponential in the depth of f. We note that these results are optimal in terms of the size of the adversarial coalition achievable in each of these settings.

Optimal Round-Complexity Results (Sections 6, 7). Finally, we obtain new protocols with low round complexity for general functions in various MPC settings. We believe that numerous results can be derived using our techniques. For concreteness, we focus on achieving perfect malicious security with optimal round complexity (i.e. 3 round). We then consider the setting of 2 round protocols, where malicious security is not achievable, and instead show statistical security with selective aborts, and (assuming one way functions) computational security with aborts. To this end, we devise round-efficient protocols for degree-2 functions with different malicious security guarantees and derive the following corollaries.

- **Fully Malicious Security in Three Rounds (Section 6).** For all f, there exists a 3 round protocol which is secure against fully malicious adversarial coalitions containing less than 1/4 fraction of the parties. For NC^1 the protocol is perfectly secure and for arbitrary polynomial-time functions the protocol has computational security with black-box access to one-way functions.

  In both cases the protocol provides *full security* (i.e. no abort). This is the optimal round complexity, as Gennaro et al. [12] showed that full security cannot be achieved in less than three rounds even when the adversary is allowed to corrupt at most 2 players. Prior to our work, it was known that general 3-round MPC with perfect security can be achieved with security threshold of $t = \alpha n$ for some small (unspecified) constant $\alpha \ll 1/4$ [11]. Our protocol shows that the threshold can be improved to $n/4$.

– **Security with Aborts in Two Rounds (Section 7).** For all f, there exists a 2 round protocol (not requiring broadcast) which is secure with *selective aborts*[5] against any adversarial coalition containing a minority of the parties. For NC^1 the protocol is statistically secure and for arbitrary polynomial-time functions the protocol has computational security with black-box access to one-way functions. This improves over the result of Ishai et al. [17], that achieve, in the same setting, security against an adversary that corrupts less than 1/3-fraction of the parties.
  We further show that in the computational setting (for polynomial-size functions) the protocol can be modified to be secure with (unanimous) *aborts*[6] at the expense of using a broadcast channel. A result with similar parameters was already shown by Ananth et al. [1]. Recently [20] showed that selective abort is the best possible security for two-round protocols that only use secure channels. Concurrently and independently from our work, Ananth et al. [2] presented a two-round protocol achieving statistical security with (unanimous) abort. Contrary to our work, they do not propose a general framework and do not achieve our general degree-2 completeness theorem or our results in the fully malicious setting.

1.2 Technical Overview

We now provide a high level overview of our techniques.

Our Master Theorem. Recall that we want to show how to encode an arbitrary protocol Π by an oracle aided protocol $\hat{\Pi}$ that uses a quadratic oracle h, while preserving the security properties of Π. As mentioned above, our techniques extend those of [3] to the malicious setting.

In [3], the authors consider the boolean circuit induced by an execution of the protocol Π, with wires corresponding to the internal values computed by all parties throughout the protocols, and gates that represent either local computation performed by a certain party, or a transmission of a value from one party to another. Their encoding constitutes of an information-theoretic point-and-permute garbled circuit [4,16,22] for this induced circuit. The encoding of Π is a protocol where the parties jointly compute this garbled circuit using their inputs and local randomness.

[5] Security with selective aborts is a notion where in the ideal model the adversary can prevent some of the honest parties of his choice from learning the output.

[6] Security with aborts is a notion where in the ideal model the adversary can prevent either all or none of the honest parties from receiving the output (but cannot allow only some of them to receive it). We specify “unanimous aborts” in places where there is a risk of confusion with the aforementioned notion of selective aborts.

The randomness required for computing the garbled circuit is distributed between the parties in a clever way that ensures that the garbled circuit can be written as $h(\ell_1, \ldots, \ell_n)$ for a quadratic h and for values ℓ_i that only depend on the local input and randomness of each party. This allows to derive a protocol encoding theorem for the semi-honest setting, where parties in $\hat{\Pi}$ simply compute their local ℓ_i and send these values to the h oracle. Garbled circuit security ensures that *any* adversarial coalition can only learn from the garbled circuit their respective views in an honest execution of Π (assuming that the ℓ_i values were computed correctly).

However, the aforementioned approach relies on the values ℓ_i being computed honestly. In contrast, a malicious adversary in this $\hat{\Pi}$ can compute the ℓ_i values belonging to the parties under its control arbitrarily, and thus a-priori we are not guaranteed that $h(\ell_1, \ldots, \ell_n)$ is even a garbled circuit at all, not to mention that it does not reveal "forbidden" values to the adversary.

Our main insight is that if the garbled circuit and the manner of distributing randomness between parties are properly defined, such a malicious modification must lead to $h(\ell_1, \ldots, \ell_n)$ being a secure garbled circuit, but one that does not encode an honest execution of Π. Instead, $h(\ell_1, \ldots, \ell_n)$ can encode an execution of Π where the parties under the adversary's control may deviate from the protocol. In other words, any cheating strategy in the compiled protocol $\hat{\Pi}$ (i.e. some adversarial modification of the values ℓ_i controlled by the adversary) translates into some cheating strategy against Π with the same adversarial coalition. Hence, $\hat{\Pi}$ inherits the security properties of Π. More details follow.

Let us first be more specific about the "partition of work" between the local functions ℓ_i and the quadratic function h. The local function ℓ_i takes the input of the ith party x_i, and two types of random tapes which we denote $\boldsymbol{s}_i, \boldsymbol{\alpha}_i$. The function performs some (deterministic) preprocessing on $\boldsymbol{\alpha}_i$, producing $\mathsf{pre}_i(\boldsymbol{\alpha}_i)$, and outputs $(x_i, \boldsymbol{s}_i, \boldsymbol{\alpha}_i, \mathsf{pre}_i(\boldsymbol{\alpha}_i))$. Our adversary is allowed to arbitrarily modify all of these values, let us examine the effects of such modification on each of these components.

- Changing the value x_i is equivalent to selecting a different input for the ith party, which cannot be avoided in any model of secure computation.
- The random string $\boldsymbol{s}_i$ is used by h as shares for *wire keys* of the garbled circuit. The exact functionality does not matter for this outline, but the important thing is that h XORs these values among all parties. Thus, choosing $\boldsymbol{s}_i$ maliciously does not buy the adversary any leverage, since h only uses the aggregate value $(\oplus_i \boldsymbol{s}_i)$, which is uniform from the adversary's viewpoint (so long as there is at least one honest party).
- The random string $\boldsymbol{\alpha}_i$ is used to produce *mask bits* for the values of the wires in the garbled circuit. Essentially, the evaluation of a point and permute garbled circuit results in producing, for each wire of the circuit that was garbled, the value of this wire XORed with a random mask bit. Crucially, the string $\boldsymbol{\alpha}_i$ contains the mask bits for the wires of the circuit whose values party i is *allowed to see.* (The definition of the induced circuit for a protocol guarantees that there is a disjoint partitioning of wires between

the parties). Hence, an adversarial choice $\boldsymbol{\alpha}_i$ gives the adversary no leverage. Such behavior can only hurt the privacy of the adversary's own wires, and has no effect on the privacy of honest parties.

– The crux of the matter is $\mathsf{pre}_i(\boldsymbol{\alpha}_i)$. The preprocessing pre_i is in fact what allows to reduce the degree of h to 2. A malicious adversary can certainly damage these computed values and indeed effect the resulting garbled circuit. What we show next is that the damage of such malleability is controllable. To explain, we go into a little more detail about the functionality of pre_i.

  Recall that each gate in the circuit to be garbled represents either a local computation by a party or communication from one party to another. The function pre_i only computes on bits in $\boldsymbol{\alpha}_i$ that are associated with inputs of local computation gates. For each such gate $\mathsf{pre}_i(\boldsymbol{\alpha}_i)$ contains four evaluations of the gate function (say NAND, w.l.o.g), on the four possible inputs in a specific permuted order, where the permutation is determined by respective $\boldsymbol{\alpha}_i$ bits. Specifically, the permutation is obtained by taking the canonical sequence $00, 01, 10, 11$, and XOR-ing it with the respective mask bits of the input wires of that gate.

  The adversary might plug in 4 arbitrary bits instead of the correct values to be computed by pre_i, regardless of the actual values it chose for the mask bits $\boldsymbol{\alpha}_i$, and possibly depending on any other value that the adversary might have. The crucial part of our argument is to notice that any change in the preprocessing can equivalently be described as a change in the *gate function*, e.g. computing OR instead of NAND, but executing this gate on the *correct* mask bits. Once this is established, we can take a step back and notice that in fact, all that the adversary can do by corrupting its $\mathsf{pre}_i(\boldsymbol{\alpha}_i)$ values is equivalent to changing the garbled circuit from one that corresponds to an honest execution of the protocol Π, to an execution of Π where the parties that are controlled by the adversary change the functionality of their local computation gates!

We conclude that even if the ℓ_i values controlled by the adversary are maliciously corrupted, h will still output a garbled circuit which corresponds to an execution of Π, possibly with some parties behaving dishonestly (the parties corresponding to the corrupted ℓ_i values). The security of this garbled circuit (which follows from the fact that the wire keys and the mask keys for honest parties remain random, as we described above) guarantees that the parties in $\hat{\Pi}^h$ learn the exact same information as they do in an execution of Π with the respective adversary (we use a perfectly secure garbled circuit which implies that the adversary's views in the two cases are identical).[7]

[7] In fact, the adversary in Π is somewhat weaker than a full malicious adversary. First, the adversarial parties are required to have the same circuit topology as honest parties, since only gate functionality changes and not the interconnection of gates. Second, the adversary cannot adjust the behavior of party i under its control based on a message received by a different party j under its control during the execution of the protocol. We find this property quite interesting and potentially useful, although

Lastly, we note that an additional modification to the [3] approach is required in order to handle broadcast channel, i.e. the possibility of a party to send a message to all other parties, such that parties are guaranteed that the same message was sent to all. This is a useful component that aids in the design of maliciously secure protocols, and is not needed in the semi-honest setting (since parties there follow the protocol specifications, so if a party is instructed to send the same value to all others, that is what it will do). If the underlying protocol Π is one that uses broadcast, this needs to be enforced by the "induced circuit" for which a garbled circuit is computed. Fortunately this is easy to handle by generalizing the point-to-point transmission gates into fan-out gates with a single input and multiple outputs. The way such gates are garbled guarantees that it is impossible to produce an execution where the outputs are inconsistent (i.e. where different parties receive different values).

The Completeness Theorem. Applying the master theorem is, on the face of it, straightforward. Instantiating Π with a protocol that is secure in the malicious setting, should immediately imply the theorem statement, and indeed the fraction of honest parties required exactly matches that of best known malicious MPC protocols with many rounds. However, there is one caveat that requires careful consideration. The encoding theorem induces a blowup in the communication and computational complexity of the protocol $\hat{\Pi}$, which is related to the size of the (information theoretic) garbled circuit of the circuit induced by Π. In particular, the size of the garbled circuit scales exponentially with the depth. We want to argue that our reduction scales with the properties of the *target function* f to be be computed. Thus, for example, using an underlying Π whose depth is (say) n times the depth of f will incur an exponential cost in the parameters of the reduction. We thus carefully analyze existing protocols so as to guarantee that there exists Π where the depth of the induced circuit only relates linearly to the depth of the function f being evaluated.

One observation that proves very helpful is that there is no need to encode local postprocessing that takes place after all the messages has been sent. That is, given a protocol Π it is sufficient to apply our master theorem on a truncated protocol Π' in which the parties send all messages as in Π, but instead of performing the final postprocessing computation they just output their view in the execution. This modification leads to a much shallower circuit for our encoding theorem and at the same time allows to achieve the required functionality and security. Functionality is maintained since the postprocessing in the final step can be done on the output of the garbled circuit evaluation, rather than being included in the garbled circuit itself. See Section 5 for more details.

Optimal Round-Complexity Results. As explained above, these are achieved by plugging in secure protocols for evaluating degree-2 functions in various models. Such protocols are usually not made explicit in the literature (as computing degree-2 functions was not a major goal until this work). However, known techniques do seem to become monotonously more round-efficient as the degree

we do not need to exploit it to derive the consequences in the cases analyzed in this paper.

drops. We apply modifications on top of existing methods in order to reduce the round complexity to the very optimum.

Three-Round Protocols with Full Security. In Section 6 we implement MPC with full security for degree-2 functions f via the following template:

1. Each party shares each of its inputs between all the parties using a sub-protocol for Verifiable Secret Sharing (VSS).
2. Each party locally computes the degree-2 functionality f over its shares and gets a share of the outputs. To enable this computation, the underlying secret sharing scheme has to be 2-multiplicative over the binary field.
3. The parties broadcast the result (after some randomization) and apply a correction procedure for handling malformed shares.

The template can be instantiated with different ingredients (e.g., for the VSS and for the recovery step). The security and round complexity of the resulting protocol depend on the corresponding properties of the underlying building blocks.

We instantiate the above template with the standard polynomial-based Shamir secret sharing scheme [23]. Gennaro et al. [11] showed that the sharing phase of this secret sharing scheme can be perfectly realized (with full security) in 2 rounds for our security threshold. This VSS natively supports secrets that are taken from a medium-size field of size at least $n + 1$, and we show how to modify it into a binary VSS.[8] Eventually, we get a 2-round binary VSS with the guarantee that at the end of the sharing phase, the honest parties hold shares that are consistent with some binary secret, even if the dealer was malicious. We observe that for our security threshold, after the third round (in which the parties broadcast their shares of the output), the honest parties can recover the output via the standard Reed-Solomon decoding algorithm.

Two-Round Protocols with Selective Abort. Here we rely on two results from [17]. In their work, they consider a weaker notion of security, *Privacy with Knowledge of Outputs* (PKO)[9], and show that:

1. Any r-rounds protocol with PKO security for functions in NC^1 (resp. polynomial-size functions) induces a r-rounds protocol with selective abort security for functions in NC^1 (resp. polynomial-size functions).
2. Any degree-2 function can be efficiently computed in two rounds with statistical PKO security for threshold $n/2$, without a broadcast channel.

[8] In particular, we use an extension field of GF(2), and add a mechanism that forces the adversary to use binary inputs. Implementing this mechanism without increasing the round complexity is somewhat challenging, and for this, we rely on some specific properties of the [11] scheme. See Sections 6 and full version for details.

[9] Intuitively, this means that the correctness of honest parties may be violated, but the adversary is required to "know" the (possibly incorrect) outputs of the honest parties. Formally, in the ideal model, the ideal functionality first delivers the outputs of the corrupted parties to the simulator, and then receives from the simulator an output to deliver to each of the uncorrupted parties.

To complete the proof, we show that our completeness theorem maintains PKO security. This turns to be somewhat subtle since, as we observe, PKO security is not always preserved under composition. (See full version for details.)

Two-Round Protocols with Abort. Lastly we use a modification from [19] which shows how a 2-round protocol with SSA security for polynomail-size functions can be converted to a 2-round protocol with SA security of similar complexity and security guarantees, at the expense of using a broadcast channel and one way functions. The general reduction, however, involves a reduction to a functionality f' that invokes the signing algorithm of a digital signature scheme. When instantiated with an arbitrary signature scheme, computing f' results in a non black-box use of a one-way function. We observe that the transformation of [19] requires only one-time secure signatures, and therefore can be instantiated with Lamport's one-time signatures (cf. [13, Chapter 6.4.1]), in which the one-way function is used only in the key-generation and verification algorithms, but not in the signing algorithm. See full version for details.

Acknowledgements

We thank Yuval Ishai for helpful discussions, for providing us several useful pointers, and for sharing with us the full version of [11].

2 Preliminaries

In this section, we define Boolean circuits, multi-party protocols, oracle-aided protocols and security for multi-party computation. Briefly, we consider an active non-adaptive rushing adversary that may be computationally unbounded or computationally bounded (depending in the context) and, unless stated otherwise, assume a fully connected network with point-to-point private channels and a broadcast.

2.1 Boolean Circuits

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

– A *(p-ary) fan-out gate*, sometimes denoted as a *transmission gate*, that has a single input and p outputs, its functionality is to copy its input to all outputs.
– A *local gate* has two input wires and one output wire. It computes some arbitrary function $G : \{0,1\}^2 \to \{0,1\}$ (that can vary from one gate to the next).

For purposes of analysis, we define the *depth* of a p-ary transmission gate to be $\lceil \log p \rceil$, and the depth of a local gate to be 1. The depth of a circuit C is the computed by considering the cumulative depth of gates along each path from an input wire to an output wire in C, and taking the maximum among all paths.

The *size* of a circuit, m, is the number of wires in the circuit (including input and output wires). We assume a topological ordering of the wires in $[m]$.

We say that two circuit C_1, C_2 are *topologically equivalent* (or have the same topology) if they are identical, except perhaps in the functions G associated with local gates.

2.2 Functionalities and Protocols

It will be convenient to treat functionalities and protocols as finite (fixed length) objects. The infinite versions of these objects will be defined and discussed laster in Section 2.3. We continue with a formal definition.

Notation. For any set $T \subsetneq [n]$, we denote $\overline{T} = [n]/T$ where n, which denotes the number of players, will be clear from the context. For any sequence $\boldsymbol{x} = (x_1, \ldots, x_n)$ and any $S \subseteq [n]$ let $\boldsymbol{x}[S]$ denote the ordered set $\{x_i\}_{i \in S}$.

Definition 2.1 (multi-party functionality). *An n-party functionality $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ is a (possibly randomized) function that maps a sequence of n inputs $\boldsymbol{x} = (x_1, \ldots, x_n)$ to a sequence of n outputs $\boldsymbol{y} = (y_1, \ldots, y_n)$. If f sends the same output to all parties then we denote its output as a scalar, i.e. we use the shorthand $f : (\{0,1\}^*)^n \to \{0,1\}^*$ and $y = f(x_1, \ldots, x_n)$.*

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

Definition 2.2 (multi-party protocol, oracles). *An* n-party, r-round protocol *Π is a tuple of $n \cdot r$ boolean circuits $\{C_{j,i}\}_{j \in [r+1], i \in [n]}$ that correspond to the computation that party i in the protocol performs before the j-th communication round (or after the last round if $j = r+1$). Each $C_{i,j}$ (except for $j = 1$ and $j = r+1$, see below) takes n input strings, and outputs n output strings. The i'-th output of $C_{j,i}$ is the message sent from party i to party i' at round j of the protocol. If $i = i'$ then the respective output is the* state *of party i after the j-th round of communication. We therefore require that for all i, i', j the i'-th output of $C_{j,i}$ has the same length as the i-th input of $C_{j+1,i'}$. In the first round of communication $C_{1,i}$ only takes one input x_i to be interpreted as party i-th input for the protocol, and possibly an additional random tape. In the last round of communication $C_{r+1,i}$ only has one output which should be interpreted as the output of party i in the protocol, sometimes denoted y_i. We let M_i denote the collection of circuits associated with party i, i.e. $M_i = (C_{1,i}, \ldots, C_{r+1,i})$ and thus denote $\Pi = (M_1, \ldots, M_n)$. The view of the party in the protocol contains its input, randomness and all messages it received during the execution.*

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

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

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

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

2.3 Correctness and Security of Protocols

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

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

Definition 2.3 (adversaries, the real model, ideal model). *An adversary (A, T) for an n-party protocol $\Pi = (M_1, \ldots, M_n)$ consists of an interactive circuit A (sometimes called the adversarial* strategy*), and a set $T \subseteq [n]$. The parties in T (resp. $\overline{T}$) are the* dishonest *(resp.* honest*) parties.*

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

A semi-honest *adversary is one in which A executes according to the parties $\{M_i\}_{i \in T}$, and outputs some function of the views of $\{M_i\}_{i \in T}$ in the protocol as the outputs of the parties in T.*

The ordered sequence of outputs of all parties in the execution above is called the output of the real-model execution and denoted $\mathsf{REAL}_{\Pi,T,A}(\boldsymbol{x})$. The ideal-model is defined by considering the trivial non-interactive f-oracle-aided protocol Υ^f in which each party simply sends its input x_i to the f-oracle, gets the output

y_i from the oracle, and terminates with this output. For an adversary (A, T) and vector of inputs $\boldsymbol{x}$ we denote the output of the ideal-model execution by $\mathsf{IDEAL}_{f,T,A}(\boldsymbol{x})$.

In the ideal-model with selective abort *(SSA), the f-oracle first delivers the outputs of the corrupted parties to the adversary, which then can decide for each uncorrupted party whether this party will receive its output or a special abort symbol. The ideal-model with* abort *(SA) is similar to SSA except that when the adversary decides to abort,* all *of the honest parties receive a special abort symbol.*[10]

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

Definition 2.4 (correctness and security of protocols). *Let $f = \{f_\kappa\}$ be an $n(\kappa)$-party functionality and $\Pi = \{\Pi_\kappa\}$ a (possibly oracle-aided) $n(\kappa)$-party protocol. We say that Π $t(\kappa)$-securely computes f if for every probabilistic non-uniform algorithm $A = \{A_\kappa\}$ and every infinite sequence of sets $\{T_\kappa\}$ where $T_\kappa \subseteq [n(\kappa)]$ is of cardinality at most $t(\kappa)$, there exists a probabilistic non-uniform algorithm $B = \{B_\kappa\}$ and a polynomial $p(\cdot)$ so that the complexity of B_κ is at most $p(|A_\kappa|)$, such that for every infinite sequence of inputs $\{\boldsymbol{x}_\kappa\}$, the distribution ensembles (indexed by κ)*

$$\mathsf{IDEAL}_{f_\kappa,T_\kappa,B_\kappa}(\boldsymbol{x}_\kappa) \quad \textit{and} \quad \mathsf{REAL}_{\Pi_\kappa,T_\kappa,A_\kappa}(\boldsymbol{x}_\kappa)$$

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

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

Definition 2.5 (secure reductions, non-interactive reductions). *If there exists a secure h-oracle-aided protocol for computing f, we say that f is reducible*

[10] The terminology of "security with abort" and "security with selective abort" is borrowed from [17] and [19] and it corresponds to the notions of "security with unanimous abort and no fairness" and "security with abort and no fairness" from [14].

to h. If the aforementioned oracle-aided protocol is non-interactive (i.e. only consists of non-adaptive calls to h) we say that the reduction is non-interactive.

Appropriate composition theorems, e.g., [13, Thms. 7.3.3, 7.4.3], guarantee that the call to h can be replaced by any secure protocol realizing g, without violating the security of the high-level protocol for f. (In the case of computational security the reduction is required, of course, to be efficient.)

3 Building Blocks

In this section we define building blocks that rely on previous works and will be used for our master theorem in Section 4. Circuit representation of protocols is defined in Section 3.1, and our presentation of point and permute garbled circuits follows in Section 3.2.

3.1 Circuit Representation of a Protocol

Recall that a protocol $\Pi = (M_1, \ldots, M_n)$, is a sequence of interactive circuits. It will be convenient to collapse all these circuits to a single "circuit representation" of a protocol. (A similar abstraction appears in [3], but some of the details differ, e.g., the treatment of fan-out gates which are needed for handling protocol that employ broadcast.)

Informally, we consider the computation of all parties throughout the protocol as parts of one large computation. Each wire of the new circuit is associated with an index corresponding to the party in the protocol that computes this value. This includes the local computations performed by parties throughout the protocol, which are represented as gates whose inputs and outputs are associated with the party who is performing the local computation, and also message transmissions between parties, that are modeled as gates that simply copy their input to the output, where the inputs are associated with the sender and outputs are associated with the receiver.

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

Definition 3.1 (Circuit Representation of a Protocol). *The* circuit representation *of a deterministic n-party protocol Π is a pair (C, P), where C is a Boolean circuit of size m as defined in Section 2.1, and $P : [m] \to [n]$ is a mapping from the wires in C to the n parties.*

Given a protocol $\Pi = (M_1, \ldots, M_n)$, the circuit C and the mapping P are defined as follows.

1. *Recalling Definition 2.2, Π consists of a sequence of circuits $C_{j,i}$ which represent the local computation of party i before the j-th round of communication (and a final circuit $C_{r+1,i}$ for the local computation after the last round of communication), we call this the j-th computational step of the protocol.*

2. *All input wires of sub-circuits that correspond to the first step in the protocol are defined as input wires of C. All output wires of sub-circuits that correspond to the last step in the protocol are defined as outputs wires of C.*
3. *The input wires representing the input state of $C_{j,i}$ are connected to the wires representing the output state of $C_{j-1,i}$ via a unary transmission gate. That is, the state of party i in the beginning of computation step j is identical to its state in the end of computation step $j-1$.*
4. *If party i expects a message in step j from party i', then the respective output wire of $C_{j-1,i'}$ is connected to the respective input wire of $C_{j,i}$ via a unary transmission gate. If party i_0 was supposed to send some value via broadcast to multiple parties $i_1, \ldots, i_p$ then a p-ary transmission gates connects the respective output wire in C_{j-1,i_0} to the input wires in $C_{j,i_1}, \ldots, C_{j,i_p}$.*
5. *Note that by the description above, the set of wires in C is exactly the union of wires of all circuits $C_{j,i}$. The mapping P associates with party i the wires of circuits $C_{j,i}$, for all j.*

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

The following is an observation that will be useful for our construction. Essentially it says that if we switch some of the gates that belong to some party with different gates, then the resulting circuit still represents a protocol.

Lemma 3.1. *Let $\Pi = (M_1, \ldots, M_n)$ be a protocol, and let (C, P) be its circuit representation. Let $T \subseteq [n]$ be some set and let H be a subset of the local gates of T such that every gate in H belongs to a party $i \in T$. Consider a circuit C' topologically equivalent to C, which is identical to C except on the gates in H. Then (C', P) is a circuit representation of a protocol $\Pi' = (M'_1, \ldots, M'_n)$ with the same round complexity and message pattern as Π, and where $M'_i = M_i$ for all $i \notin T$.*

Proof. This follows almost by definition. Define the sub-circuits $C'_{j,i}$ of C' according to their isomorphic counterparts in C. Since only local gates belonging to parties in T are changed, it follows that $C'_{j,i} = C_{j,i}$ for all j and for all $i \notin T$. Now define the party M'_i for $i \in T$ to have the functionality that in computation step j it runs the cub-circuit $C'_{j,i}$ on its state from the previous step and incoming messages, to produce the next state and outgoing messages. By definition (C', P) is the circuit representation of Π'.

3.2 Point and Permute Garbled Circuits

We present an information theoretic variant of the point-and-permute construction of [4,22]. Our variant extends the information theoretic garble circuits of [16] to handle (p-ary) transmission gates as in Definition 3.1. In addition, we slightly modify the encoding and decoding procedures, Encode and Decode, as follows. The encoding procedure Encode is decomposed into two parts, Permute and

$\mathsf{Encrypt}$, where $\mathsf{Permute}$ shuffles the truth tables of each gate based only on the mask bits $\boldsymbol{\alpha}$ (and is independent from the randomness $\boldsymbol{s}$ that is being used to generate the gate keys) and the second part $\mathsf{Encrypt}$ (for "Table encryption") generates the encrypted gate tables (based on all the randomness and on the outcome of the first part). We also modify the decoding procedure so that it outputs the masked bits of all wires of the circuits (instead of outputting only the un-masked bits of the outputs). We begin with a detailed description of the encoding and decoding procedures, and continue by analyzing their properties.

The Construction Randomness of the encoder. The encoder $\mathsf{Encode}(C, \boldsymbol{x}; \boldsymbol{s}, \boldsymbol{\alpha})$ takes as input a circuit C with local and transmission gates, with m wires in total, as well as an input $\boldsymbol{x}$ for C and random tape consisting of two strings: a vector $\boldsymbol{\alpha} = (\alpha_j)_{j\in[m]} \in \{0,1\}^m$ of masks (one for each wire), and a vector of "wire keys" $\boldsymbol{s} = (s_j^0, s_j^1)_{j\in[m]}$. The keys of the j-th wire $s_j^0, s_j^1 \in \{0,1\}^{\omega_j}$ are of length ω_j which is defined recursively. If j is an output wire then $\omega_j = 0$. If j is an input wire of local gate whose output wire is k, then $\omega_j = 2(\omega_k + 1)$. If j is an input wire of a p-ary transmission gate whose output wires are $k_1, \ldots, k_p$ then $\omega_j = \sum_{i\in[p]}(\omega_{k_i} + 1)$. By our definition of depth, the total length of $\boldsymbol{s}$, denoted by $\omega_C = 2 \cdot \sum_{j\in[m]} \omega_j$, is polynomial in m and 2^d where d is the depth of C. Lastly, if j is an input wire for a local gate and s is one of its wire keys, we let $s[0], s[1]$ denote the first and second half of s respectively.

The encoding. We now turn to the encoding procedure, which is divided into two parts, first we compute a sequence $\boldsymbol{\Gamma}$ by running a subroutine $\boldsymbol{\Gamma} = \mathsf{Permute}(C, \boldsymbol{\alpha})$ (note that this subroutine depends only on $\boldsymbol{\alpha}$ and not on any of the other input values). Then we apply $\mathsf{Encrypt}(C, \boldsymbol{x}, \boldsymbol{s}, \boldsymbol{\alpha}, \boldsymbol{\Gamma})$, which outputs the final encoding. The procedures are described below.

- The procedure $\mathsf{Permute}(C, \boldsymbol{\alpha})$ operates as follows. For every local gate g in C, with input wires $c, d \in [m]$, compute the (ordered) set

$$\Gamma_g := \left\{\gamma_g^{\beta_c,\beta_d} := G(\alpha_c \oplus \beta_c, \alpha_d \oplus \beta_d)\right\}_{\beta_c,\beta_d\in\{0,1\}} \tag{1}$$

  where $G : \{0,1\}^2 \to \{0,1\}$ is the function that the gate g computes. Let $\boldsymbol{\Gamma}$ denote the (ordered) set $\{\Gamma_g\}_g$ for all local gates in C, output $\boldsymbol{\Gamma}$.
- The procedure $\mathsf{Encrypt}(C, \boldsymbol{x}, \boldsymbol{s}, \boldsymbol{\alpha}, \boldsymbol{\Gamma})$ operates as follows. For every gate g in C, construct its *gate table* Q_g:
  - If g is a local gate, with incoming wires c, d and outgoing wire k, the gate table of g consist of four values. For every $\beta_c, \beta_d \in \{0,1\}$, compute $Q_g^{\beta_c,\beta_d}$ by setting $\gamma := \gamma_g^{\beta_c,\beta_d}$ and computing:

$$\underbrace{Q_g^{\beta_c,\beta_d}}_{\text{"ciphertext"}} := \underbrace{(s_k^\gamma \| \gamma \oplus \alpha_k)}_{\text{"message"}} \oplus\ s_c^{\alpha_c\oplus\beta_c}[\beta_d] \ \oplus\ s_d^{\alpha_d\oplus\beta_d}[\beta_c]\ .$$

    One can view $Q_g^{\beta_c,\beta_d}$ as a *one-time pad ciphertext*, encrypted using the wire keys of the input wires.

- Transmission gates are treated analogously. That is, if g is a transmission gate g with incoming wire c and outgoing wires $k_1, \ldots, k_p$, the table of g consists of two values. For every $\beta_c \in \{0,1\}$, set $\gamma = \beta_c \oplus \alpha_c$ and define

$$Q_g^{\beta_c} := \Big((s_{k_1}^{\gamma} \| \gamma \oplus \alpha_{k_1}) \| \ldots \| (s_{k_p}^{\gamma} \| \gamma \oplus \alpha_{k_p}) \Big) \ \oplus \ s_c^{\alpha_c \oplus \beta_c} \ .$$

Finally, the encoding includes the sequence $\boldsymbol{Q}$ containing all gate table values $Q_g^{\beta_c,\beta_d}, Q_g^{\beta_c}$, as well as a sequence $\boldsymbol{\sigma}$ containing the wire keys and masked values $(s_j^{x_j}, x_j \oplus \alpha_j)$ for every input wire j.

Decoding. The decoding procedure $\mathsf{Decode}(\boldsymbol{Q}, \boldsymbol{\sigma})$ takes as input a sequence of gate tables, and pairs $(s_j, \hat{v}_j)$ for the input wires. It outputs a sequence $\hat{v}_j$ for all $j \in [m]$ by traversing the gate tables in topological order. (Here we slightly deviate from the standard convention in randomized encoding literature that the decoder outputs the unmasked values of the output wires.) This is done by traversing the circuit from the inputs to the outputs as follows. For input wires j the pair $\hat{v}_j, s_j$ is given explicitly the input. For an internal wire that is an output wire of a local gate g with incoming wires c, d, this is done by using the masked bits $\hat{v}_c, \hat{v}_d$ to select the ciphertext $Q_g^{\hat{v}_c,\hat{v}_d}$ and then decrypting (i.e., XOR-ing) it with $s_c[\hat{v}_d] \oplus s_d[\hat{v}_c]$. The recovered value is then denoted $(s_k, \hat{v}_k)$. Transmission gates are treated similarly: use the masked bit $\hat{v}_c$ of the input wire to select the ciphertext $Q_g^{\hat{v}_c}$ and then XOR it with s_c to obtain $(s_{k_i}, \hat{v}_{k_i})$ for $i = 1, \ldots, p$.

Useful Properties We first state properties of $\mathsf{Encrypt}$ that will be important for our purposes.

Proposition 3.1. *The function* $\mathsf{Encrypt}$ *has algebraic degree* 2 *when written as a polynomial over the binary field in its inputs.*

Proof. This property is straightforward from the definition, since the only non-linear components in $\mathsf{Encrypt}$ are ones that require making a selection of the form s^z, where z is some variable from $\boldsymbol{\alpha}$ or $\boldsymbol{\Gamma}$ (or a linear shift thereof), such a value can be expressed as $s^0 \oplus z \cdot (s^0 \oplus s^1)$, i.e. a quadratic function. (Note that all β values are fixed and known whenever they are used.)

The next proposition follows by definition.

Proposition 3.2. *The function* $\mathsf{Encrypt}$ *is only dependent on the topology of* C *and not on the functionality* G *of local gates.*

The following propositions (3.3, 3.4) have been proven multiple times in the garbled circuit literature (cf. [16]).

Proposition 3.3 (Efficiency). *For all* $C, \boldsymbol{x}, \boldsymbol{s}, \boldsymbol{\alpha}$, *where* C *if of depth* d *and size* m, *the computational complexity of* $\mathsf{Encode}(C, \boldsymbol{x}; \boldsymbol{s}, \boldsymbol{\alpha})$ *is a fixed polynomial in* $m, 2^d$.

For every circuit C and input $\boldsymbol{x}$, we define for all $j \in [m]$ the value v_j as the value that the wire takes when evaluating C on $\boldsymbol{x}$ (in particular for input wires, $v_j = x_j$).

Proposition 3.4 (Correctness). *For all $C, \boldsymbol{x}, \boldsymbol{s}, \boldsymbol{\alpha}$, setting $z = \mathsf{Encode}(C, \boldsymbol{x}; \boldsymbol{s}, \boldsymbol{\alpha})$, and then $\{\hat{v}_j\}_{j\in[m]} = \mathsf{Decode}(z)$, it holds that $\hat{v}_j = v_j \oplus \alpha_j$.*

In Section 3.3 we will prove the following, somewhat non-standard, simulation property of garbled circuit. (The case where W is taken to be set of output wires corresponds to the standard simulation property of information-theoretic garbled circuits.)

Proposition 3.5. *There exists a* PPT *simulator* Sim *that takes as an input a circuit C, a subset of wires W, and for every wire $j \in W$ a mask-bit/intermidiate-value pair $(\alpha_j, v_j) \in \{0,1\} \times \{0,1\}$ such that the following holds. For every C, W and $\{\alpha_j\}_{j\in W}$ and every input $\boldsymbol{x}$ the random variable*

$$\mathsf{Sim}(C, W, \{\alpha_j, v_j\}_{j\in W}),$$

where the value v_j is the value induced on the j-th wire of C by the input $\boldsymbol{x}$, is distributed identically to the random variable

$$\mathsf{Encode}(C, \boldsymbol{x}; \boldsymbol{s}, \boldsymbol{\alpha}),$$

where $\boldsymbol{s}$ is uniformly random and $\boldsymbol{\alpha} = \{\alpha_j\}_{j\in W} \cup \{\alpha_j\}_{j\notin W}$ for a uniformly random $\{\alpha_j\}_{j\notin W}$.

Recall that the outcome $\boldsymbol{\Gamma}$ of $\mathsf{Permute}(C, \boldsymbol{\alpha})$ is a vector that is indexed by the gates of C where for each gate g the entry Γ_g is a four-bit string as defined in Eq. (1). The following key lemma shows that a corruption of some entries of $\boldsymbol{\Gamma}$ corresponds to applying Permute to a corrupted version of the circuit C with the same randomness $\boldsymbol{\alpha}$.

Lemma 3.2 (Corruption Lemma). *For all $C, \boldsymbol{\alpha}$, let $\boldsymbol{\Gamma} = \mathsf{Permute}(C, \boldsymbol{\alpha})$, let H be a subset of the gates of C, and let $\boldsymbol{\Gamma}'$ be a vector for which $\Gamma'_g = \Gamma_g$ for all gates $g \notin H$. Then there exists a circuit C' which is obtained from C by (possibly) modifying only gates in H, such that $\boldsymbol{\Gamma}' = \mathsf{Permute}(C', \boldsymbol{\alpha})$. Moreover, C' can be efficiently computed based on $C, H, \{\Gamma'_g\}_{g\in H}$ and based on the values of the masked bits α_i for all wires i that enter the gates in H.*

Proof. We define C' by modifying the H gates of the circuit C as follows. For all $g \in H$, consider $\Gamma'_g = \{\gamma'^{\beta_1,\beta_2}\}_{\beta_1,\beta_2\in\{0,1\}}$. Let α_1, α_2 denote the $\boldsymbol{\alpha}$ values corresponding to the input wires of g. Define a new gate functionality $G'_g : \{0,1\}^2 \to \{0,1\}$ as

$$G'(\beta_1, \beta_2) = \gamma'^{(\beta_1,\beta_2)\oplus(\alpha_1,\alpha_2)} .$$

The property $\boldsymbol{\Gamma}' = \mathsf{Permute}(C', \boldsymbol{\alpha})$ follows by definition.

3.3 Proof of Proposition 3.5

We begin with the following standard proposition that captures the privacy property of garbled circuits. Note that our Encode procedure as defined above does not release any of the $\boldsymbol{\alpha}$ values in the clear.

Proposition 3.6 (Simulation). *Let $(C^{(1)}, \boldsymbol{x}^{(1)})$, $(C^{(2)}, \boldsymbol{x}^{(2)})$ be such that $C^{(1)}, C^{(2)}$ are topologically equivalent. Then for uniformly random $\boldsymbol{s}, \boldsymbol{\alpha}$, the random variables $\mathsf{Encode}(C^{(1)}, \boldsymbol{x}^{(1)}; \boldsymbol{s}, \boldsymbol{\alpha})$ and $\mathsf{Encode}(C^{(2)}, \boldsymbol{x}^{(2)}; \boldsymbol{s}, \boldsymbol{\alpha})$ are identically distributed.*

The following claim will be useful for deriving Proposition 3.5 below.

Claim 1. *Let $(C^{(1)}, \boldsymbol{x}^{(1)}, \boldsymbol{\alpha}^{(1)})$, $(C^{(2)}, \boldsymbol{x}^{(2)}, \boldsymbol{\alpha}^{(2)})$ be such that $C^{(1)}, C^{(2)}$ are topologically equivalent, and such that $\boldsymbol{v}^{(2)} \oplus \boldsymbol{y}^{(2)} = \boldsymbol{v}^{(2)} \oplus \boldsymbol{y}^{(2)}$. Then, considering a uniformly random $\boldsymbol{s}$, the distributions $\mathsf{Encode}(C^{(1)}, \boldsymbol{x}^{(1)}; \boldsymbol{s}, \boldsymbol{\alpha}^{(1)})$ and $\mathsf{Encode}(C^{(2)}, \boldsymbol{x}^{(2)}; \boldsymbol{s}, \boldsymbol{\alpha}^{(2)})$ are identical distributed.*

Proof. Let $\boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)}$, $\boldsymbol{s}^{(1)}, \boldsymbol{s}^{(2)}$ be uniform and independent, and define, for $i \in \{1, 2\}$, random variables $\zeta^{(i)} = \mathsf{Encode}(C^{(i)}, \boldsymbol{x}^{(i)}; \boldsymbol{s}^{(i)}, \boldsymbol{\alpha}^{(i)})$. Then by Proposition 3.6, the two random variables are identically distributed: $\zeta^{(1)} \equiv \zeta^{(2)}$. We recall that by the definition of Decode, there exists a deterministic function d s.t. $d(\zeta^{(i)}) = \boldsymbol{v}^{(i)} \oplus \boldsymbol{y}^{(i)}$. As for any deterministic function, we have $(\zeta^{(1)}, d(\zeta^{(1)})) \equiv (\zeta^{(2)}, d(\zeta^{(2)}))$, i.e. $(\zeta^{(1)}, \boldsymbol{v}^{(1)} \oplus \boldsymbol{y}^{(1)}) \equiv (\zeta^{(2)}, \boldsymbol{v}^{(2)} \oplus \boldsymbol{y}^{(2)})$. This implies that for any value of $\boldsymbol{y}^*$ it holds that $(\zeta^{(1)} | \boldsymbol{v}^{(1)} \oplus \boldsymbol{y}^{(1)} = \boldsymbol{y}^*) \equiv (\zeta^{(2)} | \boldsymbol{v}^{(2)} \oplus \boldsymbol{y}^{(2)} = \boldsymbol{y}^*)$. That is, the conditional distributions are identical.

Now set $\boldsymbol{y}^* = \boldsymbol{v}^{(1)} \oplus \boldsymbol{y}^{(1)} = \boldsymbol{v}^{(2)} \oplus \boldsymbol{y}^{(2)}$, and notice that the random variable $(\zeta^{(i)} | \boldsymbol{v}^{(i)} \oplus \boldsymbol{y}^{(i)} = \boldsymbol{y}^*)$ is distributed identically to $\mathsf{Encode}(C^{(i)}, \boldsymbol{x}^{(i)}; \boldsymbol{s}, \boldsymbol{\alpha}^{(i)})$. The claim follows.

We can now prove Proposition 3.5.

Proof (Proof of Proposition 3.5). The simulator $\mathsf{Sim}(C, W, \{\alpha_j, v_j\}_{j \in W})$ considers a circuit C' topologically equivalent to C, but such that the values on the wires W are always fixed to the respective v_j regardless of the input. This can be done by fixing some of the local gates to always output the desired values. This is possible since in $\boldsymbol{v}[\overline{W}]$, for any fan-out gate, the input and all outputs take the same value. The simulator then samples random values for $\boldsymbol{\alpha}'[\overline{W}]$, and creates $\boldsymbol{\alpha}'$ by merging them with the values $\boldsymbol{\alpha}[W]$. Finally it samples a random $\boldsymbol{s}'$ and outputs $z' \leftarrow \mathsf{Encode}(C', \mathbf{0}; \boldsymbol{s}', \boldsymbol{\alpha}')$.

Consider now $z \leftarrow \mathsf{Encode}(C, \boldsymbol{x}; \boldsymbol{s}, \boldsymbol{\alpha})$, where $\boldsymbol{s}$ is uniformly random and $\boldsymbol{\alpha} = \{\alpha_j\}_{j \in W} \cup \{\alpha_j\}_{j \notin W}$ for a uniformly random $\{\alpha_j\}_{j \notin W}$. Recall that C, C' are topologically equivalent, so have the same set of wires, and let v_j, v'_j denote the values of wire j in the executions $C(\boldsymbol{x})$ and $C'(\mathbf{0})$ respectively. We note that $\boldsymbol{\alpha}[W] = \boldsymbol{\alpha}'[W]$, and that by the definition of C' it holds that $\boldsymbol{v}[W] = \boldsymbol{v}'[W]$. Since $\boldsymbol{\alpha}[\overline{W}]$, $\boldsymbol{\alpha}'[\overline{W}]$ are uniformly random, it must be the case that $\boldsymbol{v} \oplus \boldsymbol{\alpha}$ and $\boldsymbol{v}' \oplus \boldsymbol{\alpha}'$ are identically distributed. Invoking Claim 1 concludes the proof.

4 Our Master Theorem

We show how to convert any protocol Π for computing a functionality f (irrespective of round complexity) into a non-interactive h-oracle-aided protocol $\hat{\Pi}^h$, where h is a quadratic function and while preserving the security properties of Π. A formal statement follows.

Theorem 4.1 (Master Theorem). *For every n-party protocol Π there exists an n-party non interactive oracle-aided protocol $\hat{\Pi}^h$ and an $L = \mathrm{poly}(2^d, n, m)$, where d (resp. m) is the depth (resp. size) of the circuit representation of Π (see Definition 3.1), with the following properties.*

1. ***Efficiency.*** *The communication and computational complexity of $\hat{\Pi}^h$ is at most L times larger than that of Π.*
2. ***Quadratic Oracle.*** *The oracle h is a quadratic function.*
3. ***Simulation.*** *For every strategy $\hat{A}$ acting on $\hat{\Pi}^h$, there exists a strategy A of complexity at most L times larger acting on Π, such that for all $T \subseteq [n]$ and for all $\boldsymbol{x} = (x_1, \ldots, x_n)$, the distributions $\mathsf{REAL}_{\Pi,T,A}(\boldsymbol{x})$ and $\mathsf{REAL}_{\hat{\Pi}^h,T,\hat{A}}(\boldsymbol{x})$ are identical. Furthermore, if $\hat{A}$ is semi-honest (i.e. follows the protocol) then so is A.*

Note that the simulation property also guarantees that the functionality of $\hat{\Pi}$ is the same as that of Π since the outputs of honest parties is included in the real model distribution.

Remark 4.1. It suffices to prove Theorem 4.1 only for *deterministic* protocols Π, since for a randomized protocol we can always consider Π to be the induced deterministic protocol where the parties' coins are treated as part of their input. Since our theorem quantifies over all inputs $\boldsymbol{x}$, this will also capture the case where part of the input (corresponding to the random tapes of the randomized protocol) is uniformly sampled.

Remark 4.2. Interestingly, we are able to prove the theorem using strategies A that are somewhat weaker than the most general conceivable malicious strategy, in the following sense. The colluding parties can only communicate before the execution of Π starts. That is, they cannot change their strategy in intermediate rounds of Π according to messages that were received by other parties in the collusion, however they share their initial views after seeing their inputs and before the first round begins.

In the remainder of the section we prove Theorem 4.1. We note that Lemma 3.1 and Lemma 3.2 are new observations made in this work and they constitute a fundamental part of this proof.

Proof. As explained in Remark 4.1, we may assume that Π is deterministic. Our protocol essentially computes the point-and-permute encoding (Section 3.2) of the circuit representation of the protocol Π (Definition 3.1). The oracle h will correspond to the procedure $\mathsf{Encrypt}$, and the Γ values are to be precomputed by the parties. Details follow.

Let (C, P) be the circuit representation of Π, and consider the computation $\mathsf{Encode}(C, \boldsymbol{x}; \boldsymbol{s}, \boldsymbol{\alpha})$. The protocol $\hat{\Pi}^h$ is a non-interactive oracle-aided protocol, i.e. it contains a pre-processing step where each party locally computes a message to send to the oracle, followed by an oracle response and local post-processing.

- **Preprocessing.** Each party i, on input x_i, samples a uniform vector $\boldsymbol{s}_i \in \{0,1\}^{\omega_C}$ (see the description of Encode in Section 3.2 for the definition of ω_C), and uniform values α_j for all j for which $P(j) = i$ (i.e. for all wires "belonging" to player i). Then for each local gate g that belongs to player i, the party computes the values Γ_g as in Eq. (1) (note that since g belongs to i, then i possesses all α values required for this computation). Finally, player i sends $\ell_i = (x_i, \boldsymbol{s}_i, \boldsymbol{\alpha}_i, \boldsymbol{\Gamma}_i)$ to the oracle h.
- **Oracle.** The oracle h takes all messages $\ell_i = (x_i, \boldsymbol{s}_i, \boldsymbol{\alpha}_i, \boldsymbol{\Gamma}_i)$. It concatenates all x_i into a joint input $\boldsymbol{x}$ for C, unites all $\boldsymbol{\alpha}_i$ into a vector $\boldsymbol{\alpha}$ containing a value for every wire, and unites all $\boldsymbol{\Gamma}_i$ into a single $\boldsymbol{\Gamma}$ containing a set Γ_g for every local gate g. Finally, it XORs the $\boldsymbol{s}_i$ values into a single string $\boldsymbol{s} = \oplus_i \boldsymbol{s}_i$. Note that all of these are linear operations.
  Finally it computes $z = \mathsf{Encrypt}(C, \boldsymbol{x}, \boldsymbol{s}, \boldsymbol{\alpha}, \boldsymbol{\Gamma})$ and sends (the same) z to all parties as response to their query.
- **Postprocessing.** Upon receiving z, each party i applies $\mathsf{Decode}(z)$ to obtain the sequence $\hat{v}_j$ for all $j \in [m]$. Then, for any output wire j belonging to party i, it computes $\hat{v}_j \oplus \alpha_j$ to obtain the output value (recall that for a wire j belonging to party i, the value α_j was locally generated by party i and is therefore available for postprocessing). Its output contains the collection of values on these output wires.

Properties 1, 2 in the theorem follow immediately from the properties of the point-and-permute encoding (Propositions 3.3 and 3.1). It remains to prove property 3.

Let $(\hat{A}, T)$ be an adversary for $\hat{\Pi}$. Since $\hat{\Pi}$ is non-interactive, then $\hat{A}$ only gets to choose the values $\boldsymbol{\ell}[T] = \{\ell_i\}_{i \in T}$ based on the inputs $\boldsymbol{x}[T]$, and then postprocess the oracle response z. We can further simplify and consider w.l.o.g only adversaries $\hat{A}$ that are deterministic (since our simulation is perfect and therefore holds even conditioned on any random string) and do not perform any postprocessing but instead just output z (since any postprocessing results in a deterministic function of z, thus simulating z allows to simulate any such value).

Our Simulator. Our task is to produce an adversary (A, T) for the original protocol Π with the same real-model distribution as our (deterministic, no-postprocessing) $\hat{A}$. We assume throughout that $T \neq [n]$ (i.e. there exist honest parties) otherwise the result is trivial. The adversary A first runs $\hat{A}$ on $\boldsymbol{x}[T]$ to obtain the values $\ell[T]$. Let us denote by W all wires j s.t. $P(j) \in T$, i.e. all wires that belong to parties controlled by the adversary (and $\overline{W}$ the complement set of wires), and by H all local gates that are controlled by parties in T (and $\overline{H}$ the complement set of local gates). By parsing $\ell[T]$ appropriately, we derive the values $\boldsymbol{x}'[T]$, $\boldsymbol{\alpha}[W]$ and $\boldsymbol{\Gamma}[H]$, namely all α and Γ values associated with adversarially controlled parts of C. (Note that $\boldsymbol{x}'[T]$ is not necessarily identical to $\boldsymbol{x}[T]$ since the adversary is allowed to "change its input".)

By the Corruption Lemma (3.2) we can efficiently generate a circuit C' that is topologically equivalent to C and only differs from it in local gates controlled by the adversary. By Lemma 3.1, (C', P) is a protocol representation of a protocol Π' where all M'_i for $i \notin T$ are the same as in Π, but M'_i for $i \in T$ might differ.

The adversary A now sets each party $i \in T$ to (honestly) execute the protocol Π' (i.e. the machine M'_i) using its respective input x'_i. Since for honest parties $M_i = M'_i$, we have that the parties jointly execute Π' on input $\boldsymbol{x'} = \boldsymbol{x}[\overline{T}] \cup \boldsymbol{x'}[T]$. Notice that if $\hat{A}$ is semi-honest then $\boldsymbol{\Gamma'} = \boldsymbol{\Gamma}$ and thus $C' = C$, which, in turn, implies that $\Pi' = \Pi$ and therefore A is semi-honest as well.

After the end of the execution of Π', the parties under the adversary's control do not return their prescribed output in Π'. Instead, the adversary A collects the views of all parties under its control, which correspond to the set of values $\boldsymbol{v}[W]$, i.e. the values on the W wires of C' when computed on $\boldsymbol{x'}$ (however A does not know $\boldsymbol{x}[\overline{T}]$ or any of the values $\boldsymbol{v}[\overline{W}]$). Lastly we apply the simulator from Proposition 3.5, i.e. the adversary A executes $\mathsf{Sim}(C', W, \boldsymbol{\alpha}[W], \boldsymbol{v}[W]) \to z'$ and sets the outputs of all parties in T to be z'.

Proof of Simulation. It remains to show that indeed $\mathsf{REAL}_{\hat{\Pi}^h, T, \hat{A}}(\boldsymbol{x}) \equiv \mathsf{REAL}_{\Pi, T, A}(\boldsymbol{x})$. Let us fix a value for $\boldsymbol{x}$ throughout the proof. Since we assume w.l.o.g that $\hat{A}$ is deterministic, this also fixes values for $\boldsymbol{x'}[T]$, $\boldsymbol{\alpha}[W]$, $\boldsymbol{\Gamma}[H]$, and $\{\boldsymbol{s}_i\}_{i \in T}$. Recall that $\boldsymbol{x'} = \boldsymbol{x}[\overline{T}] \cup \boldsymbol{x'}[T]$ (again a fixed value).

We start by noting that in $\mathsf{REAL}_{\hat{\Pi}^h, T, \hat{A}}$ the parties in T all output the same value z, and in $\mathsf{REAL}_{\Pi, T, A}$ they all output the same z'. Letting $\boldsymbol{y}[\overline{T}]$ denote the output of $\overline{T}$ parties in $\mathsf{REAL}_{\hat{\Pi}^h, T, \hat{A}}$, and $\boldsymbol{y'}[\overline{T}]$ denote the outputs of these parties in $\mathsf{REAL}_{\Pi, T, A}$, we conclude that our goal is to prove that $(\boldsymbol{y}[\overline{T}], z)$ is distributed identically to $(\boldsymbol{y'}[\overline{T}], z')$.

Consider the distribution $(\boldsymbol{y}[\overline{T}], z)$, and note that $z = \mathsf{Encrypt}(C, \boldsymbol{x'}, \boldsymbol{s}, \boldsymbol{\alpha}, \boldsymbol{\Gamma})$. The vector $\boldsymbol{s}$ is random since it is XOR of all parties' $\boldsymbol{s}_i$ and there exists at least one honest party that samples its $\boldsymbol{s}_i$ uniformly. The vector $\boldsymbol{\alpha}$ is the union of $\boldsymbol{\alpha}[W]$ and a uniformly sampled $\boldsymbol{\alpha}[\overline{W}]$. The vector $\boldsymbol{\Gamma}$, by Lemma 3.2, is equal to $\mathsf{Permute}(C', \boldsymbol{\alpha})$. Since $\mathsf{Encrypt}$ only cares about the topology of its input circuit (Proposition 3.2), then in fact

$$\begin{aligned} z &= \mathsf{Encrypt}(C, \boldsymbol{x'}, \boldsymbol{s}, \boldsymbol{\alpha}, \boldsymbol{\Gamma}) \\ &= \mathsf{Encrypt}(C', \boldsymbol{x'}, \boldsymbol{s}, \boldsymbol{\alpha}, \boldsymbol{\Gamma}) \\ &= \mathsf{Encode}(C', \boldsymbol{x'}; \boldsymbol{s}, \boldsymbol{\alpha}) \ , \end{aligned}$$

where the last inequality is because Encode by definition first generates $\boldsymbol{\Gamma} = \mathsf{Permute}(C', \boldsymbol{\alpha})$, and then applies $\mathsf{Encrypt}$.

Defining z in this way will allow us to show that the marginal distributions of $\boldsymbol{y}[\overline{T}]$ and $\boldsymbol{y'}[\overline{T}]$ are both identical and in fact *fixed* (having fixed $\boldsymbol{x}$, deterministic $\hat{A}$). To see this, first note that by Proposition 3.4 (correctness of garbled circuit), $\boldsymbol{y}[\overline{T}]$ is determined by the values of the output wires belonging to $\overline{T}$ parties in the evaluation of C' on $\boldsymbol{x'}$. These values are determined by $C', \boldsymbol{x'}$ regardless of randomness. Likewise, $\boldsymbol{y'}[\overline{T}]$ by definition is the output of the honest parties during the execution of Π' on $\boldsymbol{x'}$, and since Π' is represented by C', the output values are exactly the output values of C'. Lastly, $z \equiv z'$ since by Proposition 3.5

$$\mathsf{Encode}(C', \boldsymbol{x'}; \boldsymbol{s}, \boldsymbol{\alpha}) \equiv \mathsf{Sim}(C', W, \boldsymbol{\alpha}[W], \boldsymbol{v}[W])$$

where the randomness is taken over $\boldsymbol{s}$, $\boldsymbol{\alpha}[\overline{W}]$ and the coins of Sim. This finalizes the proof of the theorem.

5 Completeness Theorems

In this section we prove that degree-2 functionalities are complete under non-interactive reductions. We say that a protocol has a security loss of L if any viable real-world adversary A can be simulated by an ideal-world adversary B whose complexity is at most L times larger than the complexity of A. We prove the following theorem.

Theorem 5.1 (Completeness of quadratic functions). *Let f be an n-party functionality computable by a circuit of size S and depth D. Then there exists a non-interactive reduction from the task of securely computing f to the task of computing a degree-2 functionality over $\mathbb{F}_2$. The reduction can take any of the following forms:*

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

The protocols are employed over synchronous network with pairwise private channels and a broadcast channel (which is our default setting). In all three settings, we require full security (in particular, the adversary cannot abort the honest players). It is well known that in this case the best achievable threshold is $\lceil (n/3) - 1 \rceil$ for perfect MPC (cf. [5]) and $\lceil (n/2) - 1 \rceil$ for statistical, or even computational, MPC [21]. Hence, the theorem achieves optimal security thresholds in all three cases.

As usual in the context of constant-round information-theoretic MPC, our information-theoretic protocols are efficient only for NC^1 functionalities.[12] Nevertheless, even for general functions, for which our perfect and statistical reductions are inefficient, the result remains meaningful since the protocols resist computationally unbounded adversaries.

See full version for proof of Theorem 5.1.

6 Perfect Three-Round MPC

In this section we obtain a 3-round protocol with full security (i.e. no abort) for general functions. Namely, we prove the following theorem.

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

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

Theorem 6.1 (Perfect 3-round MPC with threshold of quarter).

– *Every* NC^1 *functionality can be securely computed in 3 rounds with perfect security and threshold of* $t = \lceil \frac{n}{4} - 1 \rceil$.
– *Given a black-box access to a one-way function, the above extends to arbitrary polynomial-time functionalities at the expense of downgrading security to computational.*

By item 1 and item 3 of Theorem 5.1, the design of such protocols reduces to the design of a protocol with similar security properties for degree-2 functionalities. It therefore suffices to prove the following proposition.

Proposition 6.1. *Let* f *be an* n*-party functionality with complexity* S *and degree 2 (over the binary field). Then* f *can be perfectly computed in 3 rounds with security threshold of* $t = \lceil \frac{n}{4} - 1 \rceil$ *and complexity of* $\mathrm{poly}(n, S)$.

The proof of Proposition 6.1 appears in Section 6.1.

6.1 Proof of Proposition 6.1

VSS and friends. A key component in the proof is *verifiable secret sharing* (VSS) [7]. In such secret sharing schemes, even if the dealer acts maliciously while sharing the secret s, all of the honest parties end up with shares that are consistent with some secret s'. We consider the Shamir-based VSS for threshold t where $n = 4t + 1$. The VSS will be implemented over an extension field $\mathbb{F}$ of $\mathrm{GF}(2)$ of size at least $n + 1$, e.g., $\mathbb{F} = \mathrm{GF}(2^{\lfloor \log n+1 \rfloor})$. In particular, we will need 2-round protocols that realize the following functionalities with perfect security and threshold of t.

– The functionality Share_d in which a single designated party (denoted as the dealer) holds as an input a degree d univariate polynomial P over $\mathbb{F}$ (whose zero coefficient s plays the role of the secret) and all other parties have no input. The functionality delivers to the i-th party the value $s[i] = P(i)$.[13] We refer to $(s[1], \ldots, s[n])$ as a degree-d sharing of s. Note that security guarantees that for any adversarial set T of cardinality at most t, after the execution of Share_d the outputs of honest parties, i.e. $s[\overline{T}]$, lie on a single polynomial P' of degree d, and, if the dealer is honest (i.e. not in T) then $P' = P$. For every degree-bound $d \leq t$, Gennaro et al. [11] describe a 2-round n-party protocol that perfectly realizes Share_d.
– The functionality $\mathsf{Share}_{d,0}$ which is defined similarly to Share_d except that the free coefficient of the dealer's input polynomial P must be zero. This functionality will be employed with degree $d = 2t$, and we can realize it in

[13] As usual we assume that every $i \in [n]$ is associated with some public distinct field element $\alpha_i \neq 0$ and, by abuse of notation, we denote this element by i.

2 rounds with perfect security and threshold t via the following standard reduction to Share_t. The dealer decomposes her polynomial $P(Z)$ into

$$\sum_{j=1}^{t} Z^j R_j(Z) \tag{2}$$

where $R_1, \ldots, R_t$ are degree t polynomial that are chosen uniformly at random subject to the above constraint. Then the dealer shares each R_j via Share_t, and the i-th party gets $R_1(i), \ldots, R_t(i)$ and locally set his output to $i^1 R_1(i) + \cdots + i^t R_t(i)$.

– The functionality $\mathsf{BinShare}_t$ which is defined similarly to Share_t except that the free coefficient of the dealer's input polynomial P must be zero or one. We realize this functionality in 2 rounds with perfect security and threshold t via a reduction to the Share_t protocol of Gennaro et al. [11]. This reduction, described in the full version, is non-black-box and it relies on some concrete properties of the protocol. (To the best of our knowledge this reduction has not appeared in the literature.)

Given the above ingredients the protocol is quite straightforward. In particular, we rely on the following two standard properties of polynomial-based secret sharing: (1) *2-multiplicative*: If the parties share the secrets $(s_1, \ldots, s_k)$ via a degree-d sharing then, for every degree-2 mapping f over $\mathbb{F}$, we can get a degree $2d$-sharing of the secret $f(s_1, \ldots, s_k)$ by locally applying the degree-2 mapping f to the shares of each party. (2) *Noisy interpolation*: Given N points $(y_1, \ldots, y_N) \in \mathbb{F}^N$ with the promise that there exists a degree-D polynomial P for which $P(i) = y_i$ for all but $\lfloor (N-D)/2 \rfloor$ of $i \in [N]$, we can efficiently recover the polynomial P (and this polynomial is unique) via the standard Reed-Solomon decoder.

The protocol. Let f be a degree-2 n-party functionality. We view f as a formal degree-2 polynomial over $\mathbb{F}$ with 0-1 coefficients. For ease of notation, assume that each party holds a single input x_i, and that the functionality has a single output that is delivered to all parties. (The protocol can be easily modified to handle the more general case.)

1. In parallel, every party $i \in [n]$ that holds an input $x_i \in \{0,1\}$ samples a random degree-t polynomial P_i over $\mathbb{F}$ whose free coefficient is x_i. The party invokes the 2-round protocol that implements $\mathsf{BinShare}_t$ as a dealer whose input is P_i. All parties receive the shares $(x_i[1], \ldots, x_i[n])$. In addition, every party $i \in [n]$ chooses a random degree-$2t$ polynomial R_i whose free coefficient is zero and distribute it to all the parties using via $\mathsf{Share}_{2t,0}$.
2. Each party j computes f over its shares, i.e. $f(x_1[j], \ldots, x_n[j]) \to y[j]$. It then randomizes the result by adding the value $R_1(j) + \cdots + R_n(j)$ and broadcasts the randomized share $\tilde{y}[i]$.
3. Each party interpolates a degree $2t$ polynomial Y which is consistent with at least $n-t$ of the points $(\tilde{y}[1], \ldots, \tilde{y}[n])$, the party outputs the value $Y(0)$.

Standard analysis (cf. [8, Section 2.2]) shows that the above protocol perfectly computes f with threshold t. □

7 Two-Round MPC with Abort

We move on to the case of two-round protocols. As already mentioned, even if a broadcast channel is given we cannot hope for full security (or even fairness) when more than a single party is corrupted [12]. We therefore consider two standard relaxations of security with abort. As explained in Section 3, both notions are formalized by modifying the ideal-model in a way that grants the adversary additional power. We repeat the definition for the convenience of the reader.

– *Security with Selective Abort* (SSA) allows the adversary to selectively abort some of the honest parties (after the adversary learns his output). Formally, the ideal functionality first delivers the outputs of the corrupted parties to the simulator, which then can decide for each uncorrupted party whether this party will receive its output or a special abort symbol.
– *Security with Abort* (SA) allows the adversary to abort the honest parties even after the adversary learns his output. This is formalized similarly to SSA except that when the adversary decides to abort, *all* the honest parties receive a special abort symbol.

In the remainder of this section we prove the following theorems.

Theorem 7.1 (2-Round MPC with selective abort).

– *Every* NC^1 *functionality can be computed in 2 rounds with statistical security, selective abort and security threshold of* $t = \lceil \frac{n}{2} - 1 \rceil$. *The protocol does not use a broadcast channel.*
– *Given a black-box access to a one-way function, the above extends to arbitrary polynomial-time functionalities at the expense of downgrading security to computational.*

Theorem 7.2 (Computational 2-Round MPC with abort). *Given a black-box access to a one-way function, every polynomial-time functionality can be computed in 2 rounds with computational security, standard abort and security threshold of* $t = \lceil \frac{n}{2} - 1 \rceil$.

Theorem 7.2 and the computational part in Theorem 7.1 both introduce 2-round protocols with black-box access to one-way functions for polynomial-time functions and the same security threshold. They differ, however, since the protocols in Theorem 7.2 guarantee the stronger security notion (SA) at the expense of using a broadcast channel. (Indeed, the proof of Theorem 7.2 relies on Item 2 in Theorem 7.1.). By [20], selective abort is the best possible security for 2-round protocols that only use secure channels.

For proofs of Theorems 7.1, 7.2, see full version.

References

1. P. Ananth, A. R. Choudhuri, A. Goel, and A. Jain. Round-optimal secure multi-party computation with honest majority. In H. Shacham and A. Boldyreva, editors,

Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II, volume 10992 of *Lecture Notes in Computer Science*, pages 395–424. Springer, 2018.

2. P. Ananth, A. R. Choudhuri, A. Goel, and A. Jain. Two round information-theoretic mpc with malicious security. Cryptology ePrint Archive, Report 2018/1078, 2018. https://eprint.iacr.org/2018/1078.
3. B. Applebaum, Z. Brakerski, and R. Tsabary. Perfect secure computation in two rounds. In A. Beimel and S. Dziembowski, editors, *Theory of Cryptography - 16th International Conference, TCC 2018, Panaji, India, November 11-14, 2018, Proceedings, Part I*, volume 11239 of *Lecture Notes in Computer Science*, pages 152–174. Springer, 2018. Full version in https://eprint.iacr.org/2018/894.
4. D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols (extended abstract). In H. Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 503–513. ACM, 1990.
5. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In J. Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10. ACM, 1988.
6. F. Benhamouda and H. Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In Nielsen and Rijmen [18], pages 500–532.
7. B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pages 383–395. IEEE Computer Society, 1985.
8. I. Damgård and Y. Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In V. Shoup, editor, *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 378–394. Springer, 2005.
9. S. Garg and A. Srinivasan. Garbled protocols and two-round MPC from bilinear maps. In C. Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 588–599. IEEE Computer Society, 2017.
10. S. Garg and A. Srinivasan. Two-round multiparty secure computation from minimal assumptions. In Nielsen and Rijmen [18], pages 468–499.
11. R. Gennaro, Y. Ishai, E. Kushilevitz, and T. Rabin. The round complexity of verifiable secret sharing and secure multicast. In J. S. Vitter, P. G. Spirakis, and M. Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 580–589. ACM, 2001.
12. R. Gennaro, Y. Ishai, E. Kushilevitz, and T. Rabin. On 2-round secure multiparty computation. In M. Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 178–193. Springer, 2002.
13. O. Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications.* Cambridge University Press, 2004.
14. S. Goldwasser and Y. Lindell. Secure computation without agreement. In D. Malkhi, editor, *Distributed Computing, 16th International Conference, DISC*

2002, Toulouse, France, October 28-30, 2002 Proceedings, volume 2508 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2002.

15. Y. Ishai and E. Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 294–304. IEEE Computer Society, 2000.
16. Y. Ishai and E. Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In P. Widmayer, F. T. Ruiz, R. M. Bueno, M. Hennessy, S. Eidenbenz, and R. Conejo, editors, *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, volume 2380 of *Lecture Notes in Computer Science*, pages 244–256. Springer, 2002.
17. Y. Ishai, E. Kushilevitz, and A. Paskin. Secure multiparty computation with minimal interaction. In T. Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*, pages 577–594. Springer, 2010.
18. J. B. Nielsen and V. Rijmen, editors. *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*. Springer, 2018.
19. A. Paskin-Cherniavsky. *Secure Computation with Minimal Interaction.* PhD thesis, Technion Israel Institute of Technology, 2012.
20. A. Patra and D. Ravi. On the exact round complexity of secure three-party computation. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, pages 425–458, 2018.
21. T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In D. S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 73–85. ACM, 1989.
22. P. Rogaway. *The Round-Complexity of Secure Protocols.* PhD thesis, MIT, 1991.
23. A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.
24. A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.