Secure Protocol Transformations

Yuval Ishai* Eyal Kushilevitz† Manoj Prabhakaran‡ Amit Sahai§ Ching-Hua Yu¶

May 10, 2016

Abstract

In the rich literature of secure multi-party computation (MPC), several important results rely on "protocol transformations," whereby protocols from one model of MPC are transformed to protocols from another model. Motivated by the goal of simplifying and unifying results in the area of MPC, we formalize a general notion of black-box protocol transformations that captures previous transformations from the literature as special cases, and present several new transformations. We motivate our study of protocol transformations by presenting the following applications.

- Simplifying feasibility results:
  - Easily rederive a result in Goldreich's book (2004), on MPC with full security in the presence of an honest majority, from an earlier result in the book, on MPC that offers "security with abort."
  - Rederive the classical result of Rabin and Ben-Or (1989) by applying a transformation to the simpler protocols of Ben-Or et al. or Chaum et al. (1988).
- Efficiency improvements:
  - The first "constant-rate" MPC protocol for a constant number of parties that offers full information-theoretic security with an optimal threshold, improving over the protocol of Rabin and Ben-Or;
  - A fully secure MPC protocol with optimal threshold that improves over a previous protocol of Ben-Sasson et al. (2012) in the case of "deep and narrow" computations;
  - A fully secure MPC protocol with near-optimal threshold that improves over a previous protocol of Damgård et al. (2010) by improving the dependence on the security parameter from linear to polylogarithmic;
  - An efficient new transformation from passive-secure two-party computation in the OT-hybrid and OLE-hybrid model to zero-knowledge proofs, improving over a recent similar transformation of Hazay and Venkitasubramaniam (2016).

Finally, we prove the *impossibility* of two simple types of black-box protocol transformations, including an unconditional variant of a previous negative result of Rosulek (2012) that relied on the existence of one-way functions.

1 Introduction

Secure multi-party computation (MPC) is one of the central topics around which modern cryptography has been shaped. Research in MPC has led to major innovations in cryptography, including effective definitional

*Technion, Haifa, Israel. yuvali@cs.technion.il
†Technion, Haifa, Israel. eyalk@cs.technion.il
‡University of Illinois, Urbana-Champaign. mmp@cs.illinois.edu
§University of California, Los Angeles. sahai@cs.ucla.edu
¶University of Illinois, Urbana-Champaign. cyu17@cs.illinois.edu

approaches (e.g., simulation-based security [18, 17]), powerful and vastly applicable algorithmic techniques (starting with secret-sharing [30] and garbling schemes [33]), sharp impossibility results (e.g., [9]) and even several cryptographic concepts ahead of their time (like fully-homomorphic encryption [31]). Significantly, in recent years, some of these results have started moving from theory to practice, spurring significant further theoretical and engineering effort to optimize their performance and usability.

Over 35 years of active research, MPC has grown into a rich and complex topic, with many incomparable flavors and numerous protocols and techniques. Indeed, just cataloguing the state of the art results is a non-trivial research project in itself, as exemplified by the recent work of Perry et al. [27], which proposes classifying the existing protocols using 22 dimensions.

This diversity of models and questions forms a wide spectrum of possible tradeoffs between functionality, security, and efficiency, which partially explains the massive amount of research in the area. But this diversity also poses the risk of misdirected research efforts. For instance, if a new technique is introduced in order to obtain an efficiency improvement in one model, it is not clear a-priori to which other models the same technique may apply; and even when the same technique directly applies to other models, one typically needs to manually modify protocols and their analysis to ensure it.

While developing and maintaining a systematic database like the one in [27] is certainly helpful, we propose a complementary approach to taming the complex landscape of MPC protocols. Our approach is to relate the various flavors of MPC problems to each other by means of general *protocol transformations*. More concretely, our work studies the following high level question:

To what extent can results in one MPC model be "automatically" transformed to other models?

This question is motivated by the following goals.

- *Simplicity.* The current proofs of the main feasibility results in the area of MPC are quite involved, and results for different models share few common ingredients. We would like to obtain a simpler and more modular *joint* derivation of different feasibility results from the 1980s [34, 17, 2, 7, 28, 26], which were originally proved using very different techniques.

- *Efficiency.* Despite a lot of progress on the efficiency of MPC, there are still significant gaps between the efficiency of the best known protocols in different models. For instance, viewing the number of parties n as a constant, n-party protocols that offer full-security (with guaranteed output delivery) against $t < n/2$ malicious parties [28, 10] are asymptotically less efficient compared to similar protocols with security against $t < n/3$ parties [2], or even to protocols that offer "security with abort" against $t < n$ malicious parties [23].

A classical example of a general protocol transformation is the well known "GMW compiler," [17], which transforms any MPC protocol that offers security against passive corruptions into one that offers security against active corruptions, with the help of zero-knowledge proofs. Considering that this transformation has been behind several subsequent feasibility results, one may legitimately consider that *the GMW transformation is as important as – if not more important than – the GMW protocol itself is, as an object of study.* More recent examples include the IKOS transformation using "MPC-in-the-head" [21] and the IPS transformation that combines player-virtualization with "watchlists" [23]. Common to all these techniques is the idea that they generically transform any set of protocols that are secure for some ("easier") flavors of MPC into a protocol that is secure for another ("harder") flavor.

While these previous results demonstrate the plausibility of general MPC protocol transformations in some interesting cases, they are still far from covering the space of all desirable transformations between different MPC models and leave open several natural questions.

In this work, we initiate a systematic study of such MPC protocol transformations. We define a framework to formalize these transformations, and present a few positive and negative results. We are interested in obtaining conceptually simpler alternative proofs for known feasibility results by means of new transformations, as well as in obtaining new results. We now discuss the goals of this research in more detail.

The main theoretical motivation for studying protocol transformations is that they highlight the *essential new challenges* presented in a harder flavor of MPC compared to an easier flavor. For instance, the GMW-transformation distilled out verifying claims in zero-knowledge as the essential challenge in moving from semi-honest security to security against active corruption. As another example, in this work, we present a new transformation, that can recover the classical feasibility result of Rabin and Ben-Or [28] regarding security with guaranteed output delivery with an honest majority, from two simpler feasibility results (both of which were solved in [2, 7]): (i) security against passive corruption with an honest majority and (ii) security with guaranteed output delivery but only with an arbitrarily large fraction of honest parties. We identify achieving an intermediate security notion – security with partially identifiable abort – as the key challenge in this transformation.

As noted above, another important motivation behind studying protocol transformations is the possibility of *efficiency improvements*. On the face of it, protocol transformations are not ideal for obtaining *efficient* protocols, as one can hope to obtain extra efficiency by engineering fine details of the protocols as applicable to the specific flavor of MPC. While that may indeed be true, a protocol transformation can leverage advances in one flavor of MPC to obtain efficiency improvements in another flavor. As it turns out, this lets us obtain several *new asymptotic efficiency results* based on a single new transformation. Considering that efficiency of MPC is a well-studied area, obtaining several new result at once illustrates the power of such transformations.

There are other practical and theoretical motivations that led to this work, which we mention below.

- From a pragmatic point of view, understanding the connections across flavors of MPC will help in *modular implementations* of protocols. Indeed, the implementation of a transformation from one flavor to another would tend to be significantly simpler than an entire protocol in the latter flavor, specified and implemented from scratch.
- Roles of important techniques can often be *encapsulated as transformations* among appropriate intermediate security notions (e.g., "player elimination" can be encapsulated as implementing a transformation from "identifiable-abort-security" to full-security). In the absence of such abstraction, these techniques remain enmeshed within more complex protocols, and may not benefit from research focus that a transformation can attract.
- More generally, transformations are important in *reducing duplicated research effort*. For instance, if a new technique is introduced in order to obtain an efficiency improvement in one model, it is not clear *a priori* to which other models the same technique may apply; and even when the same technique directly applies to other models, one typically needs to manually modify protocols and their analysis to ensure it. On the other hand, if generic transformations are available across models, techniques can be easily adapted across models.
- Finally, a theoretical framework is necessary to understand the *limitations of protocol transformations*, via formal impossibility theorems. Indeed, without a rigorous notion of "black-box" transformations, it is not clear how to rule out the possibility of a "transformation" which simply discards the protocol it is given and builds one from scratch. This is especially the case for unconditional security, where the standard notions of black-box use of computational assumptions are not helpful in differentiating a legitimate transformation from one which builds its own (unconditionally secure) protocol from scratch.

A Motivating Example. As an illustration of the use of protocol transformations in simplifying the landscape of MPC protocols, we consider two protocol schemes from Goldreich's book [16, Chapter 7]. The first one obtains (stand-alone) security-with-abort against arbitrary number of corruptions by an active, probabilistic polynomial time (PPT) adversary[1] (under standard cryptographic assumptions), for general function evaluation, in a model with broadcast channels only. The second one obtains full-security (i.e., guaranteed output delivery) in the same setting, but restricting the adversary to corrupt less than half the parties. Both these protocol schemes are obtained using the GMW transformation. However, *the latter feasibility result does not take advantage of the former*, but instead uses verifiable secret-sharing (VSS) and several other techniques to achieve full-security, while retaining certain elements from the previous construction.

We point out that in fact, one could avoid the duplicated effort by giving a protocol transformation from the former flavor to the latter flavor of MPC. For this, we abstract out a slightly stronger security guarantee provided by the first protocol: while it allows an adversary to abort the protocol after learning its own input, aborting always leads to identification of at least one party that is corrupted by the adversary. This notion of security is often referred to as security with identifiable-abort [22]. In Section 4.1, we show that one can easily transform such a protocol into a protocol with full-security.

Security Augmentation and Efficiency Leveraging. Typically, an MPC protocol transformation falls into one of two broad (informally defined) classes: *security augmentation* and *efficiency leveraging*. Security augmentation refers to building MPC protocols with strong security guarantees by transforming MPC protocols with weaker security guarantees. The IPS compiler (see Appendix B) is an instance of security augmentation. Efficiency leveraging, on the other hand, aims to improve the efficiency of MPC protocols, without necessarily increasing their security guarantee. In such a transformation, the original (inefficient) protocol will typically be used on a "small" sub-computation task, in combination with other cheaper (but less secure) protocols applied to the original "large" computation task. The goal of the sub-computation task is usually to ensure that the strong attacks on the final protocol has the effect of weak attacks on an execution of the cheaper, less secure protocol. An instance of efficiency leveraging is given by Bracha's transformation [4], in which the strength of the security guarantee corresponds to the corruption-threshold (i.e., what fraction of parties are corrupted) that can be tolerated.

1.1 Our Contributions

Framework. Firstly, we formalize the notion of a Black-Box Transformation (BBT) from protocol schemes satisfying some security (or efficiency) requirements to a protocol scheme satisfying some other requirements.[2] Towards this, we formalize notions like protocol schemes (which map functionalities to protocols) and security definitions (which are just sets of pairs of functionalities and protocols), all in a fairly abstract fashion. A BBT itself is modeled using a circuit that describes a protocol's structure as a program built from various components.

The framework is general enough to cast all of the above mentioned transformation (GMW, Bracha, IKOS and IPS) as instances of BBT.

We remark that we treat security notions highly abstractly, and do not impose any conditions on how security is proven. However, in all our positive results and examples, security definitions use a simulation

[1] One may consider static or adaptive corruption here. By default, we shall consider adaptive adversaries in all constructions in this paper.

[2] The term "Black-Box" refers to the fact that (the next-message function of) the resulting protocol uses (the next-message function of) all the constituent protocols and the functionality itself as oracles; however, note that the constituent protocols themselves may depend on their functionalities in a non-black-box manner.

paradigm, and one could define a "fully" blackbox transformation by requiring that the simulator of the protcol resulting from the transformation be constructed in a black-box manner from the simulators of the given protocols. For the sake of simplicity, and to keep the focus on the structure of the constructions rather than on the proofs of security, we do not formally include this restriction in our definition of BBT. We also point out that this strengthens our impossibility results.

New Transformations and Consequences. We present a new transformation which can be used to obtain known and new results about (information-theoretically) secure MPC for general function evaluation, with guaranteed output delivery, given an honest-majority and a broadcast channel. Our transformation yields such an MPC scheme starting from two protocol schemes – one achieving full-security, but for a lower threshold (βn corruption threshold, for some $\beta > 0$) and one achieving semi-honest security under honest-majority (Corollary 9). (See the next section for an overview of the transformation, and the various intermediate transformations that lead to it.) From this transformation we obtain the following results:

1. We readily obtain the result of Rabin and Ben-Or [28] as a consequence of the earlier work of Ben-Or et al. and Chaum et al. [2, 7], via the above transformation.

2. We obtain the first "constant-rate" MPC protocol scheme with guaranteed output delivery against corruption of less than $n/2$ parties, provided the number of parties is constant (Corollary 10). That is, the total communication in this protocol is at most $c_n|C|$, where C is the circuit representation of the function, and c_n is a constant independent of the security parameter of C, that depends only on the number of parties. This result is obtained – following the lead of [23][3] – by applying our transformation to the scheme of [13] (combined with a secret-sharing scheme due to [8]) and the semi-honest secure scheme of [2].

3. Next, we present an *efficiency leveraging* transformation, which is designed to improve the efficiency of a protocol scheme with full-security, by combining it with a (cheaper) protocol which achieves security-with-abort (Theorem 8). By applying this transformation to the above protocol with full-security and an efficient protocol with security-with-abort from [15], we obtain a "scalable" MPC protocol with full-security and optimal corruption-threshold – i.e., tolerating corruption of less than $n/2$ parties (Corollary 11).[4] For an arguably natural class of functions (namely, sequential computations, where the size of a circuit implementing the function is comparable to its depth), this is the first scalable protocol with full-security and optimal threshold (complementing a result of [3], which obtains similar efficiency for circuits which are of relatively low depth).

4. We present an efficient new transformation from two-party protocols in the OT-hybrid or OLE-hybrid model that offer security against passive corruptions to zero-knowledge proofs in the commitment-hybrid model, improving over a recent similar transformation of Hazay and Venkitasubramaniam [19]. (We note that the IKOS transformation for protocols in such hybrid models requires at least 3 parties.) The transformation from [19] cannot be applied in the OLE-hybrid model, and when applied to natural protocols in the OT-hybrid model such as the GMW protocol, it requires several separate commitments for each gate in the circuit. Our transformation for the OLE-hybrid model can lead to practical zero-knowledge proofs for *arithmetic* circuits and in both hybrids our transformation requires just a constant number of

[3] In [23], these two protocol schemes were combined to obtain a similar constant-rate protocol, but in the oblivious-transfer (OT) hybrid model and with security-with-abort.

[4] Here the term "scalable" denotes that for evaluating large circuits C, the *communication complexity per party* scales as $\tilde{O}(|C|)$ (up to polylog multiplicative factors and polynomial additive terms of the security parameter and the number of partiesh).

commitments overall (for a constant soundness error). This transformation may have relevance to the recent line of work on practical zero-knowledge proofs initiated in [25].

5. Our final application considers the problem of relaxing the corruption threshold from the optimal $n/2$ to $n(1/2 - \epsilon)$, for any constant $\epsilon > 0$. In this case, we obtain a *highly scalable protocol* in which the *total* communication for evaluating a circuit C is $\tilde{O}(|C|)$, ignoring additive terms that depend on the number of parties, but not the size of the circuit (Corollary 12). This improves over a result of [14].[5]

   For this, we apply Bracha's transformation [4] to one of the above protocols. Specifically, we use Bracha's transformation to combine an outer protocol that has a relatively low corruption threshold but is highly scalable with respect to communication and computation (in our case the one from [14]), and an inner protocol with optimal threshold (in our case, the one from item 2 above), to obtain a protocol with a near-optimal threshold.

Impossibility Results. One may ask if security against active corruption can solely be based on security against semi-honest adversaries. Such questions can be formalized as questions about the existence of a BBT. We present two impossibility results:

1. We consider the question of functionally-black-box protocol schemes, introduced by Rosulek [29]. (This is a special case of protocol transformations where no protocol scheme is provided to the transformation.) Rosulek demonstrated a two-party functionality family for which there is no functionally black-box protocol, *assuming the existence of one-way functions*. We present an unconditional version of this result (Theorem 1).

2. We show a functionality family – namely, zero-knowledge proof functionalities – for which there is no BBT from semi-honest security to security (with abort) against active adversaries (Theorem 2).

We remark that the proof of our second result breaks down if we expanded the family of functionalities from ZK functionalities to all efficient functionalities. We leave it as an important open problem to prove broader impossibility results for *general* computation (in which the family considered is the family of all functionalities).

1.2 Technical Overview

Black-Box Transformations. We make precise a notion of a black-box transformation among protocol schemes. Given a functionality f, a black-box transformation can define new functionalities (which are syntactically just programs) that access f in a black-box manner. Then, it can invoke a given protocol scheme on any such functionality, to obtain a protocol (which is, again, a program). The transformation can repeat these steps of defining new functionalities in terms of programs it already has, and of invoking given protocol schemes on such functionalities any number of times. At the end, it outputs one of the programs as its protocol.

Example: IPS Transformation. An example of a black-box transformation (that we shall build on later) is the IPS transformation [23]. We shall graphically represent a transformation using a circuit diagram like the one in Figure 1.

[5] In [14], in the absence of broadcast channels, the near-optimal threshold of $n(\frac{1}{3} - \epsilon)$ was considered. We can extend our result to this setting by implementing broadcast channels among a constant number of parties, with a constant factor blow-up in communication.

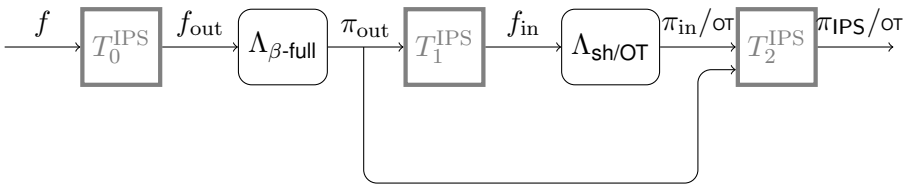

Figure 1: Black-Box Transformation in the IPS compiler

Here, each rectangular node (labeled T_0^{IPS}, T_1^{IPS} and T_2^{IPS}) outputs a program which makes black-box access to one or more programs input to that node. T_0^{IPS} converts an n-party functionality f into a functionality f_{out} involving n "clients" and N "servers". T_1^{IPS} defines f_{in} to be an n-party functionality in which the trusted party carries out the program of a server in the protocol π_{out}. The bulk of the compiler is part of the transformation T_2^{IPS}, which combines the programs of two protocols π_{out} and π_{in} in a black-box way to define the final protocol.

The diagram also shows two other nodes, labeled $\Lambda_{\beta\text{-full}}$ and $\Lambda_{\mathsf{sh/OT}}$, each of which take as input a functionality (f_{out} and f_{in} resp.) and produces a protocol (π_{out} and π_{in} resp.). The labels on the nodes indicate the security guarantees required of these protocols (security against active corruption of strictly less than a $\beta > 0$ fraction of the parties, and security against semi-honest corruption, in the OT hybrid model resp.). [23] show that irrespective of what protocol schemes are used to define the protocols produced by these nodes, as long as those schemes meet the required security conditions, the resulting protocol will be a protocol for f with security against active corruption of any number of parties.

New Transformations. We present several new transformations, some of which are summarized in Table 1. In particular, we show how to transform a low-threshold fully-secure protocol scheme and a high/optimal-threshold semi-honest secure protocol scheme to a high/optimal-threshold protocol with full-security (presented as Corollary 9). The main step is to achieve a weaker notion of security (called "security with partially-identifiable-abort") against the same high fraction of corruption. Then, we show how a protocol with partially-identifiable-abort security can be transformed to one with full-security.

The second of these two transformations turns out to be easy, using "Error-Correcting Secret-Sharing" or ECSS (also known as robust secret-sharing) [6], which can be realized easily using ordinary Secret-Sharing and one-time message authentication codes (MAC) (see Appendix D). Partially-identifiable-abort-security allows us to perform, in case of an abort, a *player elimination* process, so that an honest majority is maintained. By carrying this out not on the original function, but on a function which accepts ECSS-shared inputs and produces ECSS-shared outputs, we show how to obtain full-security. The more challenging transformations is obtaining partially-identifiable-abort-security in the first place, as discussed below.

Obtaining Partially-Identifiable-Abort Security. This transformation is based on the IPS transformation [23] which, however, was not designed for the setting with an honest majority. Hence, it relied on an OT-hybrid model, and could obtain only "security with abort." We modify this transformation in a couple of

[6]Note that a naïve protocol which runs π_1 first and in the event of an abort, runs π_2 for the same functionality does not work. If π_1 aborting is considered as an abort event, then it gives the same efficiency guarantee, but is not an id_α-secure scheme, because if π_2 completes without an abort, the protocol fails to identify an α-corrupt set. If π_1 aborting is not considered an abort event, the protocol fails to meet the efficiency guarantee.

From	To	Theorem	Notes
id_α-security, $t < \alpha n$	full, $t < \alpha n$	Theorem 3, Theorem 4	Using player-elimination. Theorem 4 relies on a non-blackbox decomposition of the function, and yields efficiency close to the non-abort-case efficiency of the given protocol.
(sh-security, $t < \alpha n$) and (full-security, $t < \beta n$)	id_α, $t < \alpha n$	Theorem 5, Theorem 6	An honest-majority version of the IPS transformation. Any $\beta > 0$ suffices. Theorem 6 saves a factor of n using an expander graph-based watchlist scheme.
(sh-security, $t < \alpha n$) and (full-security, $t < \beta n$)	full, $t < \alpha n$	Corollary 9	Combining the above two.
(abort-secure π_1, $t < \alpha n$) and (id_α-secure π_2, $t < \alpha n$)	id_α, $t < \alpha n$	Theorem 7	Efficiency Leveraging: resulting protocol almost as efficient as π_1 when there is no abort.[6]
(abort-secure π_1, $t < \alpha n$) and (full-secure π_2, $t < \alpha n$)	full, $t < \alpha n$	Theorem 8	Efficiency Leveraging: resulting protocol is almost as efficient as π_1. From Theorem 7 and Theorem 4. Relies on a non-blackbox decomposition of the function.

Table 1: A summary of the main black-box transformations in this paper. The first column lists the protocol scheme(s) given, and the second column lists the protocol scheme obtained. t stands for the number of parties that can be corrupted. id_α-security denotes partially-identifiable-abort security, in which, in the event of an abort, a set of parties, at least α fraction of which are corrupt, is identified by all honest parties. sh-security stands for security against semi-honest corruption, abort and full-security stand for security against active corruption, with the latter having guaranteed output delivery.

ways to obtain partially-identifiable-abort security in the honest-majority setting, in the plain model (with a broadcast channel). There are two major modifications we introduce, summarized below.

Watchlist Channels in the Plain Model. An important aspect of the IPS transformation is a collection of "watchlist channels" used by each party to monitor secretly chosen instances of a semi-honest secure inner protocol. In the IPS transformation, Rabin OT is used to implement the watchlist channel. Instead, we rely on a weaker variant, $\widetilde{\mathsf{OT}}$, which we can directly implement in the honest-majority setting (without even broadcast channels), using Shamir's secret-sharing. $\widetilde{\mathsf{OT}}$ allows an adversary to selectively cause aborts when there is no erasure. The reason this suffices for building a watchlist channel is that this functionality will be applied to random inputs, and when an abort occurs, we can safely identify a pair of inconsistent parties – at least one of which is corrupt – by having all parties reveal their views in the protocol (over a broadcast channel).[7]

Obtaining Partially-Identifiable Abort Instead of Abort. In the original IPS transformation, even if the outer protocol has security with guaranteed output delivery, the final protocol offers only security with abort (without any identification of the corrupt parties). This is due to the fact that when a party detects an inconsistency, it simply aborts the protocol. In the setting with honest majority, we show how to modify the IPS transformation, so as to obtain partially-identifiable abort, such that a set of two parties can be identified of which at least one is guaranteed to be corrupt.

[7] When no abort occurs, the adversary can indeed learn some information (i.e., that an erasure occurred), but this can happen only in a small number of instances before an abort occurs.

Consider when P_i detects an inconsistency in the messages reported over a watchlist channel that it has access to, in an inner protocol session. In this case, P_i cannot exactly identify the source of inconsistency, but only localize it to a pair of parties P_{i_1}, P_{i_2}, one of which is corrupt. However, since P_i itself could be a corrupt party, at this point the honest parties can agree on one of (P_i, P_{i_1}, P_{i_2}) being corrupt. But being able to identify a set in which only $1/3$ fraction is guaranteed to be corrupt falls below our required guarantee of 1 out of 2 being corrupt.

To further localize corruption, we require all the parties to broadcast their views in the inner-protocol session in which an inconsistency was detected, as they had earlier communicated over the watchlist channel to P_i. If an inconsistency is detected among the broadcast views, then all parties can identify a pair (P_{i_1}, P_{i_2}) which are inconsistent with each other. On the other hand, if all the views that are broadcast are consistent with each other, then, if P_i had indeed observed an inconsistency earlier, it can point out one party P_{i_1} which reported a view over the watchlist channel different from the one it reported over the broadcast channel. Then P_i is required to broadcast this party's identity, and all parties agree on the pair (P_i, P_{i_1}).

To see that this transformation retains security, note that by causing an abort, the adversary can cause at most one server's computation to be revealed over the broadcast channel. This corresponds to the adversary corrupting one extra server in the outer protocol. Since the choice of parameters in the IPS compiler leaves a comfortable margin for the number of server corruptions, this does not affect the overall security.

Efficiency Improvements. When considering a non-constant number of parties, there are a couple of major sources of inefficiency in the transformation above, which we can address.

Firstly, in the transformation from partially-identifiable-abort security to full security, the protocol could be restarted $\Theta(n)$ times. To avoid this overhead, we require the function to be given in the form of a composition of $\Theta(n)$ functions (for instance, a layered circuit with $\Theta(n)$ layers), each one of approximately the same size complexity. Then, one can restrict the duplicated effort for each restart to correspond to a single component, and can ensure that overall $O(n)$ restarts can only about double the cost.

Secondly, in the IPS compiler, every party can potentially watch every inner protocol session. This requires that all the communication in each inner-protocol session is sent out (encrypted with one-time pads) to all the n parties. To avoid this overhead, we can use an expander graph to define which parties may watch the execution of which servers. Specifically, we can use an expander graph between the set of parties and the set of servers in the outer protocol, in which *the degree of each server is a constant*, but any subset of $n/2$ parties has in its neighborhood (i.e., will potentially watch) almost all of the servers. Thus, the communication in each inner-protocol session (corresponding to the servers in the outer protocol) is sent out to only a constant number of parties.

Efficiency Leveraging: Transformations for Improving Efficiency. We present a new instance of efficiency leveraging, in which an MPC protocol scheme with full-security is "extended" by leveraging the efficiency of cheaper MPC protocols which only offer security with abort. Specifically, we show how to combine a protocol which guarantees only security with abort given an honest majority (e.g., from [15]) and a protocol with full-security given honest majority (like the one we constructed above) to obtain one which approaches the efficiency of the former protocol while enjoying full-security like the latter.

The basic idea is simple. We can obtain a protocol with $1/2$-identifiable-abort security as follows: given a functionality, we will run a protocol with security-with-abort to compute it; if the protocol terminates without aborting (as confirmed with the help of broadcast messages), then our protocol terminates successfully. If it aborts, then we run an (inefficient) MPC protocol with full-security for a functionality which accepts the views in the first protocol and detects a pair of parties with conflicting views, at least one of which is

corrupt (if no conflict is detected, then a party who aborted in the first place can be identified as a corrupt party, since, as part of the security guarantees, we shall require zero probability for abort if all parties run honestly). To make this idea work, we need to ensure that the inefficient MPC is called only on a small piece of computation. With appropriate parameters for decomposition of the function, this indeed gives new asymptotic results (for relatively "narrow" circuits).

Negative Results. We prove two negative results. Firstly, we show that there is a function family $\mathcal{F}$ such that there is no "functionally blackbox" protocol scheme [29] for $\mathcal{F}$ (even for semi-honest security). The family $\mathcal{F}$ consists of boolean functions of the form f_α, where $\alpha \in \{0,1\}^k$ and $f_\alpha(x,y) = 1$ if and only if $x \oplus y = \alpha$.

Our second negative result shows a function family $\mathcal{G}$ such that semi-honest secure protocol schemes for $\mathcal{G}$ cannot be converted in a blackbox manner to protocols with active security (with abort). We choose $\mathcal{G}$ to be the family of zero-knowledge proofs for a class of relations. Then, there is a semi-honest secure protocol for $\mathcal{G}$ which only accesses the given functionality $f \in \mathcal{G}$ in a blackbox manner. Hence, a blackbox transformation from semi-honest secure protocol schemes to schemes with active security translates to a functionally blackbox protocol scheme for $\mathcal{G}$ with active security.

To complete the proof, we show how to define $\mathcal{G}$ (assuming the existence of a pseudorandom function) such that there is no active secure, functionally blackbox protocol scheme for $\mathcal{G}$.

1.3 Organization of the Paper

The rest of the paper is organized as follows. Section 2 includes several basic definitions of the framework (with more details in Appendix A), and Section 3 defines the notion of a blackbox transformation. In Section 4, we give some simple transformations, including a new transformation that improves on a recent result by [19]. Section 5 presents two impossibility results regarding blackbox transformations. Section 6 through Section 8 present several transformations, which are summarized in Table 1. Section 9 presents the results we obtain by applying these transformations to protocol schemes in the literature.

2 Preliminaries

The basic objects in our framework are *protocols*. Technically, a protocol is specified by a single program (say, Turing Machine) for the "next-message function" of all the parties in the protocol. In Appendix A, we formal define protocols (Definition A.1). We shall write Π to denote the set of all protocols.

A *functionality* is technically just a special instance of a protocol, involving a trusted party. We often abuse our notation and refer to the trusted party's program as the functionality. We shall often refer to a *functionality family* $\mathcal{F}$, which is simply a set of functionalities, i.e., $\mathcal{F} \subseteq \Pi$. We denote the family of all probabilistic polynomial time computable secure function evaluation functionalities by $\mathcal{F}^*$ (represented by circuits).

We use a *synchronous model* of communication (with rushing adversaries), so that all parties in a protocol proceed in a round-by-round fashion. Note that this is applicable to ideal functionalities too. However, typically we are not interested in the exact number of rounds in the ideal functionality, as long as it finishes within a polynomial number of rounds.

$\Lambda^{\mathcal{F}}_{\mathsf{secure}}$	(f,π) s.t. $f \in \mathcal{F}$ and π meets the definition secure (for a polynomial-round version of f). If $\mathcal{F} = \mathcal{F}^*$, the family of all probabilistic polynomial time function evaluation functionalities, we simply write $\Lambda_{\mathsf{secure}}$.		
α-secure	secure, restricted to corruption of strictly less than α fraction of the parties.	secure/F	protocol is in the F-hybrid model. e.g., secure/BC denotes protocols using broadcast channels.
sa	standalone security (default is UC security).	ppt	adversary is PPT (default is unbounded adversary).
sh	semi-honest adversary.	full	active adversary (with guaranteed output delivery).
abort	adversary may learn its output and then decide which honest parties get their outputs and which do not.	id_θ	same as abort, but on abort, honest parties agree on a non-empty set of parties, at least a θ fraction of which is corrupt. We shall abbreviate α-id_α as α-id.

Table 2: Terminology used for guarantees from protocols.

2.1 Security Definitions

Technically, a *security definition* for a functionality family $\mathcal{F}$ is formalized as a relation $\Lambda \subseteq \mathcal{F} \times \Pi$. The intention is that $(f,\pi) \in \Lambda$ iff π is a secure protocol for f. For a security notion named secure, the corresponding relation will typically be written as $\Lambda_{\mathsf{secure}}$.

In Table 2 we name some of the main security definitions considered in our results. For instance, $\Lambda^{\mathcal{F}}_{\alpha\text{-full/BC}}$ includes all pairs (f,π) such that f is a functionality in the family $\mathcal{F}$, and π is a UC-secure protocol with guaranteed output delivery (within a polynomial number of rounds), against computationally unbounded adversaries who may adaptively corrupt strictly less than α fraction of the parties, and BC means that the protocol uses a broadcast channel. In all our security notions, for simplicity of our transformations, we require that an honest party aborts the protocol only if there is no possible honest execution of the protocol that is consistent with its view. We also define a security notion generalizing the notion of security with identifiable abort:

Security with θ-Identifiable Abort. We define a functionality $f^{\langle \mathsf{id}_\theta \rangle}$ to formalize the notion of security with θ-identifiable abort.

Given a normal form functionality f (see Appendix A), functionality $f^{\langle \mathsf{id}_\theta \rangle}$ is a normal form functionality which internally runs f and interacts with Adv as follows.

1. Accept the inputs from all parties (including honest parties and parties corrupted by Adv) and forward to f. (If there is no input from P_i, substitute it with a dummy input.) Set the output vector as set by f.
2. If Adv sends getoutput, then send the corrupted parties' outputs to Adv.
3. If Adv sends $(\mathsf{corrupt}, T)$ s.t. T is a subset of parties in which at least a θ fraction are corrupt, then change the output of all honest parties to be $(\mathsf{corrupt}, T)$.
4. **Output phase:** Deliver the (current) output to all parties.

2.2 Protocol Schemes

A *protocol scheme* maps a functionality to a protocol (with a desired security property).

Definition 2.1 (Λ-scheme)**.** $\mathcal{P} : \mathcal{F} \to \Pi$ *is said to be a* Λ-scheme *if* $\mathcal{F}$ *is a functionality family such that* $\Lambda \subseteq \mathcal{F} \times \Pi$*, and for every* $f \in \mathcal{F}$*,* $(f, \mathcal{P}(f)) \in \Lambda$*.*

For example, the semi-honest BGW-protocol scheme is a $\Lambda^{\mathcal{F}}_{\alpha\text{-sh}}$-scheme where $\mathcal{F}$ is the family of all circuit-evaluation functionalities and $\alpha = \frac{1}{2}$. Typical protocol schemes are *uniform*, in that there is a Turing Machine which, on input a standardized description of f, for $f \in \mathcal{F}$, outputs the code of $\mathcal{P}(f)$.

2.2.1 Complexity Notation.

To discuss asymptotic efficiency guarantees of protocol schemes, we augment the notation for security definitions to include protocols' communication (and sometimes, computational) cost. Typically, a protocol's complexity is measured as a function of some complexity measure of the functionality f that it is realizing, as well as the number of parties n and the security parameter k of the protocol execution. For each functionality family, we shall require a cost measure $\mathsf{size} : \mathcal{F} \to \mathbb{Z}^+$, that maps $f \in \mathcal{F}$ to a positive integer. We stress that a functionality f denotes a specific implementation (of a trusted party in a protocol), and so there can be different $f \in \mathcal{F}$ which are all functionally equivalent, but with differing values of $\mathsf{size}(f)$.

To capture the typical efficiency guarantees in the literature, we define a p - $\Lambda^{\mathcal{F}}_{\mathsf{secure}}$ scheme as a $\Lambda^{\mathcal{F}}_{\mathsf{secure}}$ scheme $\mathcal{P}$ such that for any $f \in \mathcal{F}$, $\mathcal{P}(f)$ is a protocol whose communication cost (for n parties, and security parameter k) is

$$O(p(n,k) \cdot \mathsf{size}(f) + \mathrm{poly}(n,k)). \tag{1}$$

For typical functionality families $\mathcal{F}$, a functionality $f \in \mathcal{F}$ is represented as a circuit C_f, and $\mathsf{size}(f)$ is the size of C_f. The function $p(n,k)$ reflects the multiplicative overhead of secure computation, on top of the size of the (insecure) computation.

Often, protocol schemes which offer a smaller value for $p(n,k)$ incur additive costs. To denote protocol schemes with such complexities, we use a more detailed notation: $(p,q,r;\mathsf{D})$ - $\Lambda^{\mathcal{F}}_{\mathsf{secure}}$ schemes are $\Lambda^{\mathcal{F}}_{\mathsf{secure}}$ schemes $\mathcal{P}$ such that for all $f \in \mathcal{F}$, the communication cost of $\mathcal{P}(f)$ is $O(p(n,k) \cdot \mathsf{size}(f) + \mathrm{poly}(n,k) \cdot \mathsf{D}(f))$, its *computation cost* is $O(q(n,k) \cdot \mathsf{size}(f) + \mathrm{poly}(n,k) \cdot \mathsf{D}(f))$, and its *randomness cost* is $O(r(n,k) \cdot \mathsf{size}(f) + \mathrm{poly}(n,k) \cdot \mathsf{D}(f))$. Here D is a secondary cost measure – typically the depth of the circuit C_f – which is often much smaller than $\mathsf{size}(f)$. We omit D to indicate that $\mathsf{D}(f)$ is a constant and omit q and/or r to leave them as unspecified $\mathrm{poly}(n,k)$ functions. We omit $\mathcal{F}$ if it equals $\mathcal{F}^*$, the family of all probabilistic polynomial time function evaluation functionalities.

For functionality families using circuit representation, a traditional choice for D is depth: $\mathsf{depth}(f)$ denotes the depth of the circuit C_f representing f. We shall find it useful to define another function width, defined as follows. For any topological sorting of the gates in the circuit, define a sorted-cut as a partition of the gates into two sets so that all the gates in one part appear before any gate in the other part, in the topologically sorted order; the max-sorted-cut for a sort order is the maximum number of wires crossing a sorted-cut. $\mathsf{width}(f)$ is the value of the max-sorted-cut of C_f minimized over all topological sorts of C_f. (Alternately, we could require the topological sort to be part of the circuit specification. In this case, an appropriate model of computation would be a *linear bijection straight-line program* [1], and width would correspond to the number of "registers" in the program.)

For protocol schemes providing partially-identifiable security, like α-id-schemes, we sometimes want to distinguish the cost of an execution without an abort event and that with an abort event (and identification): a $\langle \gamma, \delta \rangle$ - $\Lambda_{\alpha\text{-}\mathsf{id}}$ scheme denotes a $\Lambda_{\alpha\text{-}\mathsf{id}}$ scheme $\mathcal{P}$ such that the communication cost of $\mathcal{P}(f)$ is $O(\gamma(n,k) \cdot \mathsf{size}(f) + \mathrm{poly}(n,k))$ without abort events and $O(\delta(n,k) \cdot \mathsf{size}(f) + \mathrm{poly}(n,k))$ with abort.

Finally, we write $(p,q,r;\mathsf{D})\sim\Lambda^{\mathcal{F}}_{\mathsf{secure}}$ instead of $(p,q,r;\mathsf{D})$ - $\Lambda^{\mathcal{F}}_{\mathsf{secure}}$ and so on, if we intend to use $\widetilde{O}(\cdot)$

instead of $O(\cdot)$ in the above costs.[8] The notation is summarized in Table 3.

$(p, q, r; \mathsf{D})$ - $\Lambda_{\mathsf{secure}}$	$\Lambda_{\mathsf{secure}}$ scheme $\mathcal{P}$ s.t. the communication cost of $\mathcal{P}(f)$ is $O(p(n,k) \cdot \mathsf{size}(f) + \mathrm{poly}(n,k) \cdot \mathsf{D}(f))$, the computation cost is $O(q(n,k) \cdot \mathsf{size}(f) + \mathrm{poly}(n,k) \cdot \mathsf{D}(f))$ and randomness cost is $O(r(n,k) \cdot \mathsf{size}(f) + \mathrm{poly}(n,k) \cdot \mathsf{D}(f))$.
$(p, q; D)$ - $\Lambda_{\mathsf{secure}}$	$(p, q, r; \mathsf{D})$ - $\Lambda_{\mathsf{secure}}$, where $r(n,k)$ is $\mathrm{poly}(n,k)$.
(p, q) - $\Lambda_{\mathsf{secure}}$	$(p, q; \mathsf{D})$ - $\Lambda_{\mathsf{secure}}$, where $D(f)$ is a constant
$(p; D)$ - $\Lambda_{\mathsf{secure}}$	$(p, q; \mathsf{D})$ - $\Lambda_{\mathsf{secure}}$, where $q(f)$ is $\mathrm{poly}(n,k)$
p - $\Lambda_{\mathsf{secure}}$	$(p, q; \mathsf{D})$ - $\Lambda_{\mathsf{secure}}$, where $D(f)$ is a constant and $q(f)$ is $\mathrm{poly}(n,k)$
$\langle \gamma, \delta \rangle$ - $\Lambda_{\alpha\text{-id}}$	$\Lambda_{\mathsf{secure}}$ scheme $\mathcal{P}$ s.t. the communication cost of $\mathcal{P}(f)$ is $O(\gamma(n,k) \cdot \mathsf{size}(f) + \mathrm{poly}(n,k))$ without abort events and $O(\delta(n,k) \cdot \mathsf{size}(f) + \mathrm{poly}(n,k))$ with abort.
(params)~$\Lambda_{\mathsf{secure}}$	Similar to (params) - $\Lambda_{\mathsf{secure}}$ scheme, but with $\widetilde{O}(\cdot)$ instead of $O(\cdot)$.

Table 3: Additional notation for protocol schemes (for n parties, and security parameter k).

2.3 Error-Correcting Secret-Sharing

Some of our transformations rely on a simple variant of secret-sharing that has been referred to as robust secret-sharing or as honest-dealer VSS [28, 11, 6]. To clarify the nature of this primitive, we shall call it *Error-Correcting Secret-Sharing (ECSS)*, and define it formally below. For the sake of completeness, an elementary construction of an ECSS scheme is given in Appendix D.

Definition 2.2 (Error-Correcting Secret Sharing)**.** *A pair of algorithms* $(\mathsf{share}, \mathsf{reconstruct})$ *is said to be an* (n, t)*-Error-Correcting Secret Sharing (ECSS) scheme over a message space* $\mathcal{M}$ *if the following hold:*

1. **Secrecy:** *For all* $s \in \mathcal{M}$ *and* $N_c \subseteq [n], |N_c| < t$*, the distribution of* $\{\sigma_i\}_{i \in N_c}$ *is independent of* s*, where* $(\sigma_1, ..., \sigma_n) \leftarrow \mathsf{share}(s)$.

2. **Reconstruction from upto** t **erroneous shares:** *For all* $s \in \mathcal{M}$*, and all* $(\sigma_1, ..., \sigma_n)$ *and* $(\sigma'_1, ..., \sigma'_n)$ *such that* $\Pr[(\sigma_1, ..., \sigma_n) \leftarrow \mathsf{share}(s)] > 0$ *and* $|\{i \mid \sigma'_i = \sigma_i\}| \geq n - t$*, it holds that* $\mathsf{reconstruct}(\sigma'_1, ..., \sigma'_n) = s$.

3 Defining Black-Box Transformations

In this section, we present our framework of black-box transformations, which operates on protocol schemes (Definition 2.1). More specifically, a black-box transformation defines a Λ-scheme in terms of Λ'-schemes, for one or more other security notions Λ'. We present our definition in two parts – first the syntax of a transformation, followed by its security requirements.

Definition 3.1 (Black-Box Transformation (BBT): Syntax)**.** *A* BBT for a functionality family $\mathcal{F}$ *is defined as a circuit* C *with*

- *a single input wire taking a functionality* $f \in \mathcal{F}$*,*

[8] $\widetilde{O}(h)$ denotes $O(h \cdot \mathrm{polylog} h)$.

- *a single output wire outputting a protocol* $\pi \in \Pi$,
- *one or more* black-box nodes *labeled with oracle TMs* $T_1, \cdots, T_s$,
- *one or more* protocol nodes *labeled with relations* $\Lambda_1, \cdots, \Lambda_t$ *where* $\Lambda_i \subseteq \mathcal{F}_i \times \Pi$ *for some functionality family* $\mathcal{F}_i$.

For a black-box node labeled with T_i *we require that the number of oracles accessed by* T_i *is equal to the number of input wires to that node. For a protocol node, we require that there is only one input wire.*

Given such a circuit C and protocol schemes $\mathcal{P}_1, \cdots, \mathcal{P}_t$ such that each $\mathcal{P}_i$ is a Λ_i-scheme, we define $C^{\mathcal{P}_1,\ldots,\mathcal{P}_t}(f) \in \Pi$ as follows. We shall set the value on each wire in C to be a protocol in Π (possibly a functionality), starting with the input wire and ending with the output wire, which is taken as the value $C^{\mathcal{P}_1,\ldots,\mathcal{P}_t}(f)$. First, set the value on the input wire to be f. Then, for any black-box node with all its input wires' values already set to values $\pi_1, \cdots, \pi_d$, set its output wire's value to $T_i^{\pi_1,\cdots,\pi_d}$, where T_i is the label on the node. For any protocol node with its input wire's value set to π, set its output wire's value to $\mathcal{P}_i(\pi)$, where i is the index of the protocol node in C (if $\mathcal{P}_i(\pi)$ is undefined, then $C^{\mathcal{P}_1,\ldots,\mathcal{P}_t}(f)$ is undefined).

Definition 3.2 (Black-Box Transformation (BBT))**.** *We say that a BBT* C*, for a functionality family* $\mathcal{F}$*, is a* BBT from $\{\Lambda_1, \cdots, \Lambda_t\}$ to Λ*, if* C *has* t *protocol nodes labeled with* $(\Lambda_1, \cdots, \Lambda_t)$ *and, for all* $f \in \mathcal{F}$ *and all* $(\mathcal{P}_1, \cdots, \mathcal{P}_t)$ *such that each* $\mathcal{P}_i$ *is a* Λ_i*-scheme, we have* $(f, C^{\mathcal{P}_1,\ldots,\mathcal{P}_t}(f)) \in \Lambda$.

4 Examples of Black-Box Transformations

In Appendix B, we illustrate how several important constructions from the literature are in fact BBTs from simpler security notions or simpler function families, to more demanding ones. This list includes Bracha's compiler [4] (from high-threshold (and low-efficiency) security and low-threshold (and high-efficiency) security to a high-threshold (and high-efficiency) security), the IKOS compiler [21] (from semi-honest secure MPC and and honest-majority secure MPC to active security for Zero-Knowledge proofs) and the IPS compiler [23] (as above, but for arbitrary MPC). The GMW compiler [17] could also be viewed as a BBT (from semi-honest security and active security specialized to zero-knowledge functionality, to active security).

It is helpful to visualize these transformations using "circuit diagrams." An example of the IPS transformation was given in Figure 1. Similar diagrams for the other examples mentioned above are given in Appendix B.

Below we discuss two new simple BBTs, which yield much simpler alternatives to more complex constructions in the literature.

Improving Over [19]. Very recently, Hazay and Venkitasubramaniam [19], presented an IKOS-like transformation that starts from any (semi-honest) two-party protocol *in the OT-hybrid model* and gives a zero-knowledge proof system in the commitment-hybrid model. We present a different transformation that has several advantages over [19]: our transformation may start with a two-party protocol in the OLE-hybrid model,[9] whereas the one from [19] seems inherently restricted to the OT-hybrid model. Perhaps more importantly, to achieve a constant level of soundness our transformation uses only a constant number of

[9] OLE stands for Oblivious Linear function Evaluation. It is a generalization of Oblivious Transfer where a sender has (a, b) in a field $\mathbb{F}$ and the receiver has $x \in \mathbb{F}$. At the end of the protocol, the receiver will learn $ax + b$ while the sender learns nothing. OLE-based protocols are useful for arithmetic computation. Such protocols are obtained in [24] by generalizing the OT-based GMW protocol [17].

commitments (to long strings), compared to the protocol in [19] that uses as many commitments as the number of OT calls. For the simplest case of the GMW protocol applied to a boolean circuit of size s, our protocol requires only 6 commitments whose total length is $O(|C|)$ whereas the protocol from [19] requires $O(|C|)$ separate bit-commitments. These features of our transformation make it appealing for the design of practical ZK protocols based on OT-hybrid and OLE-hybrid protocols such as GMW.

Our transformation, as well as the IKOS transformation on which it is based, are presented in Appendix B. At a high-level, we give a simple BBT from a 2-party semi-honest MPC protocol scheme in the OLE-hybrid model to a 3-party 1-private MPC protocol scheme in the plain model; this transformation is then readily composed with the IKOS transformation (which can be applied to a 1-private protocol) to obtain our full transformation.

4.1 A Pedagogical Application

One of the results from Goldreich's textbook [16] can be simplified using a BBT. In [16], two separate protocols for $\Lambda_{\text{abort-ppt-sa-id}}$ (i.e., security-with-identifiable-abort) and $\Lambda_{1/2-\text{full-ppt-sa}}$ (i.e., security with guaranteed output delivery, with an honest majority) are presented, with the latter relying on VSS. Below, we give a BBT from $\Lambda_{\text{abort-ppt-sa-id}}$ to $\Lambda_{1/2-\text{full-ppt-sa}}$, that uses ECSS (see Section 2.3) instead of VSS.

To evaluate an n-party function f, each party shares its input using an $\lceil n/2 \rceil$-out-of-n error-correcting secret-sharing (ECSS) scheme (see Section 2.3), and sends the resulting shares to the n parties. We remark that an ECSS is much simpler than, say, a VSS protocol, and can be constructed readily by adding message authentication code (MAC) tags to the shares of any threshold secret sharing scheme (such as Shamir's scheme). Then, the parties use a protocol π from the protocol scheme with security-with-identifiable-abort to evaluate a function f', which takes shares as its inputs, reconstructs them to get inputs for f, evaluates f and reshares the outputs among all parties, again using ECSS. If the shares given as inputs have fewer than $n/2$ errors, f' can error-correct and recover the original input being shared; otherwise it defines the reconstructed value to be a default value (this corresponds to the shares not being generated correctly in the first place). If the protocol π for f' does not abort, then all the parties are expected to redistribute the shares they received from π, so that each party gets all the shares of its output; due to the error-correcting property, and since the adversary can corrupt less than $n/2$ of the shares received by each honest party, every honest party will be able to correctly recover its output. On the other hand, if the protocol π aborts, due to the identifiable-abort security guarantee, all honest parties will agree on the identity of one corrupt party. Note that at this point, even though the adversary may learn its outputs from π (i.e., outputs of f'), these carry no information and can be efficiently simulated (by a simulator running the protocol with arbitrary inputs for the honest parties). Hence, the parties can simply eliminate the identified party (and still retain honest majority), and restart the entire protocol on a smaller functionality in which the eliminated party's input is replaced by a default value. This process must eventually terminate, after at most $\lceil n/2 \rceil$ attempts, guaranteeing output for all honest parties.

An ad-hoc use of the above "player elimination" technique was made in several previous MPC protocols (see, e.g., [20] and references therein). In contrast, our use of this technique yields a *completely general transformation* from a weaker flavor of MPC to a stronger one.

5 Impossibility of Black-Box Transformations

In this section, we present some impossibility results for BBT. Before proceeding, we emphasize that in the definition of BBT, we *do not* require the security proofs to be black-box in any form. In particular, the

simulators used to define security can arbitrarily depend on the functionality in a non-black-box manner. As such, the impossibility results on BBT are of a rather strong nature.

Our first impossibility results relates to an interesting special case of a BBT, namely, BBT from $\emptyset$ to Λ. This corresponds to the notion of a *functionally-black-box* protocol introduced by Rosulek [29], wherein there is an oracle TM such that for all $f \in \mathcal{F}$, T^f is a secure protocol (according to Λ) for f. Rosulek demontrated a two-party functionality family for which there is no functionally black-box protocol, *assuming the existence of one-way functions*. We present an unconditional version of this result.

Theorem 1. *There exists a two-party functionality family $\mathcal{F}$ such that there is no BBT from $\emptyset$ to $\Lambda^{\mathcal{F}}_{\mathsf{sh}}$. In particular, there is no BBT from $\emptyset$ to $\Lambda^{\mathcal{F}^*}_{\mathsf{sh}}$.*

The detailed proof is given in Appendix C. Here we sketch the main ideas of the proof.

Proof sketch: The family $\mathcal{F}$ we shall use to prove the theorem consists of boolean functions of the form f_α, $\alpha \in \{0,1\}^k$, where $f_\alpha(x,y) = 1$ if and only if $x \oplus y = \alpha$. To show that there can be no secure protocol for f_α, in which the two parties access the function only in a blackbox manner, we consider the following experiment. Pick x, y, α uniformly and independently at random, and run the protocol for f_α with inputs x, y. Then we argue that the probability for both of the following events should be negligible:

(A) Either party queries their oracle with (p, q) such that $p \oplus q = \alpha$.

(B) Either party queries their oracle with (p, q) such that $p \oplus q = x \oplus y$.

The probability of event A is negligible since α is chosen uniformly at random, and the parties make only a polynomial number of queries. The reason for the probability of event B being negligible is the security of the protocol: in an ideal world, since $x \oplus y \neq \alpha$, a corrupt party (simulator), even given α, can learn only a negligible amount of information about the other party's input. Now, we consider a "coupled" experiment in which instead of α, we pick $\alpha^* = x \oplus y$, and run the same protocol but now for f_{α^*}. It can be argued that for the random tapes in the protocol for which events (A) and (B) does not occur in the first case, they will not occur in the second run too. Thus with high probability, both the executions produce the same output, violating the correctness of the protocol. □

Also, we consider the question of showing impossibility of BBT from semi-honest security to active security. We present such a result conditioned on the existence of one-way functions.

Theorem 2. *Assuming the existence of one-way functions, there exists a two-party functionality family $\mathcal{G}$ such that there is no BBT from $\{\Lambda^{\mathcal{G}}_{\mathsf{sh}}\}$ to $\Lambda^{\mathcal{G}}_{\mathsf{abort}}$.*

We present the intuition behind the proof below, and defer the detailed proof to Appendix C, *Proof sketch:* We will let $\mathcal{G}$ to be the family of zero-knowledge proofs for a class of relations. Then, there is a semi-honest secure protcol for $\mathcal{G}$ which only access the given functionality $f \in \mathcal{G}$ in a blackbox manner. Hence, a blackbox transform from semi-honest secure protocol schemes to schemes with active security translates to a functionally blackbox protocol scheme for $\mathcal{G}$ with active security. To show that this does not exist, we assume the existence of a pseudorandom function and define $\mathcal{G}$ based on it.

Here we briefly sketch an argument that instead uses a random oracle O to define $\mathcal{G}$. The relation R for which we define $\mathcal{G}$ consists of (x, w) such that $O(w) = x$. To show that there can be no ZK protocol for this relation in which the parties only have blackbox access to an oracle for the relation R (but the simulator may depend on O), we consider a cheating prover as follows. When given (x, w) and access to R, it uses a wrapper around R to turn it into relation which accepts (x, w) (and does not accept (x', w) for $x' \neq x$), but otherwise behaves like R. Then the cheating prover runs the honest prover with access to the modified oracle. Using the ZK property we can argue that an honest verifier, when given a random x, cannot detect

the difference between interacting with the real prover and the cheating prover. Thus, if the protocol is complete, the cheating prover will be able to break soundness. □

6 A BBT from Partially-Identifiable-Abort to Full Security

We present a simple black-box transformation from *partially-identifiable abort security* (formalized using $\Lambda_{\alpha\text{-id}}$ below) to full security. This will be an important ingredient in our applications in Section 9. First, we present a simple but general version of this transformation (which suffices for feasibility results); in Theorem 4, we shall present a more efficient variant.

Theorem 3. *For any $0 \leq \alpha \leq 1/2$, there exists a BBT from $\Lambda_{\alpha\text{-id/BC}}$ to $\Lambda_{\alpha\text{-full/BC}}$. Specifically, there is a BBT from p - $\Lambda_{\alpha\text{-id/BC}}$ to $(np; \mathsf{D})$ - $\Lambda_{\alpha\text{-full/BC}}$, where $\mathsf{D}(f)$ is the input plus output size of f.*

Our tools behind this construction are relatively simple. In particular, we do not use verifiable secret-sharing (VSS), but instead use the much simpler primitive Error-Correcting Secret-Sharing (ECSS) (see Section 2.3), which can be realized easily using ordinary Secret-Sharing and one-time message authentication codes (MAC) (see Appendix D).

Here we give a high level overview of the construction, with a complete description defered to Appendix E.1. The idea behind this BBT is that if we have a protocol which either completes the computation or identifies a set of parties such that at least α fraction of which are corrupt, then, in the event of an abort, we can remove the identified set of parties from active computation and restart the computation. Note that this preserves the corruption threshold of α (i.e., strictly less than α fraction remains corrupt) among the set of "active" parties.

For this idea to work, we need to keep the outputs secret-shared (so that by aborting, the adversary does not learn any useful information, even though it receives its outputs from the computation), and after the computation finishes, guarantee reconstruction. Further, we need to use secret-sharing to let all the parties deliver their inputs to the set of active parties. All this will be achieved using ECSS in a straightforward manner, for $\alpha \leq 1/2$.

A More Efficient Variant. In the above BBT, we restarted the entire computation in the event of an abort. To avoid this, we rely on having access to a "layered representation" of the function. Formally, consider a parametrized functionality $\hat{f}$, parametrized by an index $i \in \{1, \cdots, d\}$, such that $f = \hat{f}[d] \circ ... \circ \hat{f}[1]$, such that $\mathsf{size}(\hat{f}[i]) = O(\mathsf{size}(f)/d)$, for all i. We define $\mathsf{width}_d(f)$ to be the smallest number w such that there exists a decomposition of f into d layers, each of size $O(\mathsf{size}(f)/d)$, such that the number of output wires from any layer is at most w. We shall typically take d to be a polynomial $d(n, k)$. Note that $\mathsf{width}(f)$ defined in Section 2.2.1 is an upper-bound on $\mathsf{width}_d(f)$ for all d.

Since decomposing f into $\hat{f}$ is not a black-box operation, we require a "protocol scheme" that carries out this decomposition. For this we define a $\Lambda_{\mathsf{layer}[d]}$ scheme to be one which maps f to a parametrized function $\hat{f}$ such that

$$f = \hat{f}[d] \circ \cdots \circ \hat{f}[1],$$

and $\forall i \in [d]$, $\mathsf{size}(\hat{f}[i]) = O(\mathsf{size}(f)/d)$ and the number of bits output by $\hat{f}[i] \leq \mathsf{width}_d(f)$.

Then, as shown in Appendix E.2, we obtain the following efficiency improvement over Theorem 3.

Theorem 4. *For any $0 < \alpha \leq 1/2$, there exists a BBT from $\{\Lambda_{\mathsf{layer}[d]}, \langle \gamma, \delta \rangle$ - $\Lambda_{\alpha\text{-id}}\}$ to $(\gamma; \mathsf{D})$ - $\Lambda_{\alpha\text{-full}}$, where $d(n, k) = n \cdot \frac{\delta(n,k)}{\gamma(n,k)}$ and $\mathsf{D}(f) = \mathsf{width}_d(f)$.*

7 A BBT From $\{\Lambda_{\alpha\text{-sh}}, \Lambda_{\beta\text{-full}}\}$ to $\Lambda_{\alpha\text{-id}}$

Our goal in this section is to obtain a BBT that increases the corruption threshold of a fully secure protocol, by combining it with a semi-honest protocol which has the higher threshold. Given Theorem 3, it suffices to obtain a protocol with partially-identifiable-abort against the higher corruption threshold. Formally, we shall prove the following theorem, which is interesting when $\beta < \alpha$

Theorem 5. *For any $0 < \alpha, \beta \leq 1/2$, there exists a BBT from $\{\Lambda_{\alpha\text{-sh}}, \Lambda_{\beta\text{-full}}\}$ to $\Lambda_{\alpha\text{-id/BC}}$.*

The BBT from $(\Lambda_{\beta\text{-full}}, \Lambda_{\alpha\text{-sh}})$ to $\Lambda_{\alpha\text{-id/BC}}$, shown in Figure 8(b), resembles the IPS compiler. The main difference is that here we require an honest-majority protocol to implement the watchlist mechanism, and we need to achieve $\Lambda_{\alpha\text{-id/BC}}$ instead of $\Lambda_{\alpha\text{-abort/OT}}$. Nevertheless, the black-box transformation T_1 is identical to that in IPS, T_1^{IPS}.[10]

T_2 is obtained by some easy modifications to the corresponding transformation in the IPS compiler, T_2^{IPS}. First, we recall the structure of T_2^{IPS}, in Figure 2. We can interpret it as consisting of a "core" compiler IPS_{core}, which produces a protocol in a "watchlist-channel hybrid" model (also using OT if it is needed by the inner protocol). Separately, the watchlist-channel functionality $\mathcal{W}$ (see Appendix F.2) was realized using a protocol w_{IPS} in the OT-hybrid model. Finally, the former was composed with the latter to obtain a protocol in the OT-hybrid model.

In our case, we modify the protocol generated by IPS_{core} and the watchlist protocol, before composing them. In particular, the watchlist protocol is modified so that it realizes a functionality $\mathcal{W}^*$ (described below) which facilitates $1/2$-identification in case of abort. Further, the modified protocol does not rely on an OT-hybrid model (but on a functionality $\widetilde{\text{OT}}$ that is readily realized in the honest-majority setting using a protocol $\pi_{\widetilde{\text{OT}}}$). Similarly, the protocol generated by IPS_{core} is modified also to facilitate $1/2$-identification, and to be in the $\mathcal{W}^*$-hybrid than in the $\mathcal{W}$-hybrid. Figure 2 shows the new components in our construction, namely the protocol $\pi_{\widetilde{\text{OT}}}$, and two transformations T_{id} and T_{id}^*.

Functionality $\mathcal{W}^*$. We shall require a watchlist setup functionality with a form of partially-identifiable abort security. Recall that we can define a normal form functionality $\mathcal{W}^{\langle \text{id}_{1/2} \rangle}$ from the normal form functionality $\mathcal{W}$. $\mathcal{W}^*$ is a weaker functionality obtained by modifying $\mathcal{W}^{\langle \text{id}_{1/2} \rangle}$ so that when it aborts (and identifies a set of parties), the adversary is given the inputs of all the parties in the protocol. (Since the inputs to $\mathcal{W}$ will be random strings in our construction, it will be safe to use $\mathcal{W}^*$ instead of $\mathcal{W}^{\langle \text{id}_{1/2} \rangle}$.) A formal description of $\mathcal{W}^*$ is given in Appendix F.2, where we shall also describe the (simple) transformation T_{id}^* as well as the protocol $\pi_{\widetilde{\text{OT}}}$.

Transformation T_{id}. The goal of the transformation T_{id}, shown in Figure 2, is to give a $1/2$-identification protocol that the honest parties can carry out if an abort happens in the IPS-compiled protocol. This transformation follows the outline sketched in Section 1.2 (see paragraph *Obtaining Partially-Identifiable-Abort Security*). A formal description is given in Appendix F.3.

7.1 Using a Sparse Watchlist

The BBT in Theorem 5 is in fact a BBT from $\{(p_{\text{in}}, q_{\text{in}}, r_{\text{in}})\text{-}\Lambda_{\alpha\text{-sh}}, (p_{\text{out}}, q_{\text{out}})\text{-}\Lambda_{\beta\text{-full}}\}$ to $p\text{-}\Lambda_{\alpha\text{-id/BC}}$, where $p = n^2 \cdot (p_{\text{in}} + r_{\text{in}}) \cdot (q_{\text{out}} + n \cdot p_{\text{out}})$ (see Appendix F.4). But by exploiting the honest majority guarantee which was absent in the setting of [23], we can state the following version.

[10] See Figure 1. The functionality f_{in} is a functionality whose trusted party implements the "servers" in the protocol π_{out}.

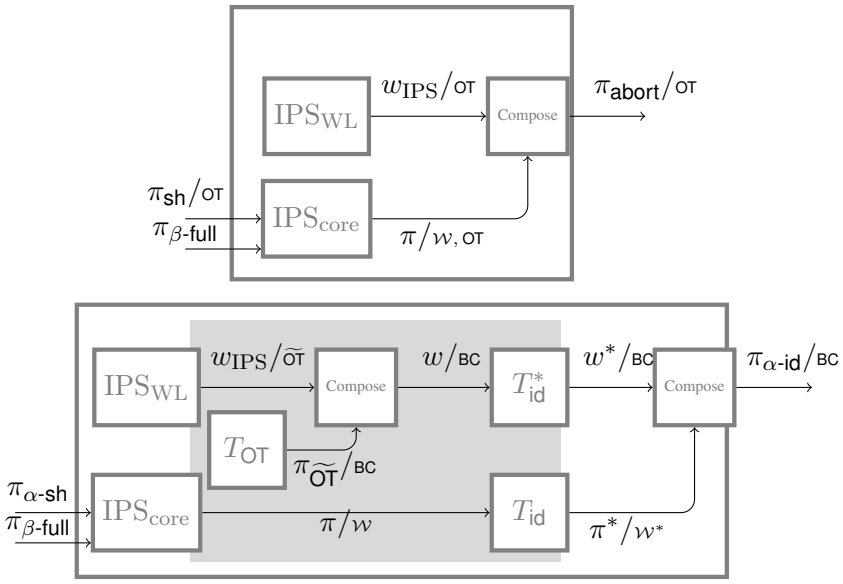

Figure 2: T_2^{IPS} and T_2. The shaded region shows the new components in T_2. Note that T_2 retains $\mathrm{IPS}_{\mathrm{core}}$ and $\mathrm{IPS}_{\mathrm{WL}}$ from T_2^{IPS} as it is.

Theorem 6. *For any $0 < \alpha, \beta \leq 1/2$, and polynomials $p_{\mathrm{in}}, q_{\mathrm{in}}, r_{\mathrm{in}}, p_{\mathrm{out}}, q_{\mathrm{out}}$, there exists a BBT from $\{(p_{\mathrm{in}}, q_{\mathrm{in}}, r_{\mathrm{in}})\text{-}\Lambda_{\alpha\text{-}\mathsf{sh}}, (p_{\mathrm{out}}, q_{\mathrm{out}})\text{-}\Lambda_{\beta\text{-}\mathsf{full}}\}$ to $p\text{-}\Lambda_{\alpha\text{-}\mathsf{id}/\mathsf{BC}}$, where $p = n \cdot (p_{\mathrm{in}} + r_{\mathrm{in}}) \cdot (q_{\mathrm{out}} + n \cdot p_{\mathrm{out}})$.*

The above result saves a factor of n compared to the previous transformation. The efficiency improvement comes from a sparser watchlist mechanism (using an expander graph to define which parties may watch the execution of which servers) in the BBT from $(\Lambda_{\beta\text{-}\mathsf{full}}, \Lambda_{\alpha\text{-}\mathsf{sh}})$ to $\Lambda_{\alpha\text{-}\mathsf{id}/\mathsf{BC}}$. We present the details in Appendix G.

8 Efficiency Leveraging

Bracha's transformation is a classical example of efficiency leveraging. It was originally proposed in the context of byzantine agreement [4], and later applied to MPC protocols (see, e.g., [14]). Below, we record a version of this result that is sufficient for our applications (see Appendix B for a graphical depiction).

Proposition 1 (Bracha's Transformation [4])**.** *Let $0 < \epsilon, \beta \leq \alpha \leq 1/2$, and let $p'(n,k) = c_n$ be independent of k. Then, for each $\mathsf{secure} \in \{\mathsf{sh}, \mathsf{abort}, \mathsf{full}\}$ and any function D, there exists a BBT from $\{(p, q; \mathsf{D})\text{-}\Lambda^{\mathcal{F}}_{\beta\text{-}\mathsf{secure}}, p'\text{-}\Lambda_{\alpha\text{-}\mathsf{secure}}\}$ to $(p''; \mathsf{D})\text{-}\Lambda^{\mathcal{F}}_{(\alpha-\epsilon)\text{-}\mathsf{secure}}$, where $p''(n,k) = p(n,k) + q(n,k)$.*

In this section, we present a new instance of efficiency leveraging for full-security: a simple BBT from $\{\Lambda_{\alpha\text{-}\mathsf{abort}}, \Lambda_{\alpha\text{-}\mathsf{full}}\}$ to $\Lambda_{\alpha\text{-}\mathsf{full}}$, in which the resulting protocol's efficiency is comparable to that of the protocol in $\Lambda_{\alpha\text{-}\mathsf{abort}}$.

First we present a efficiency leveraging transformation for $\Lambda_{\alpha\text{-}\mathsf{id}}$ which can then be combined with Theorem 4 to obtain efficiency leveraging for $\Lambda_{\alpha\text{-}\mathsf{full}}$. In our efficiency leveraging transformation for $\Lambda_{\alpha\text{-}\mathsf{id}}$ the efficiency of the resulting protocol, *when there is no abort event*, is comparable to that of a cheaper $\Lambda_{\alpha\text{-}\mathsf{abort}}$ protocol. Formally, we have the following theorem.

Theorem 7. *For any $0 \leq \alpha \leq 1/2$, and functions $p, q, p' \in \mathrm{poly}(n, k)$, there exists a BBT from $\{(p, q)\text{-}\Lambda_{\alpha\text{-abort}}, p'\text{-}\Lambda_{\alpha\text{-id}}\}$ to $\langle\gamma, \delta\rangle\text{-}\Lambda_{\alpha\text{-id}}$, where $\gamma = p$ and $\delta = p' \cdot (p + q)$.*

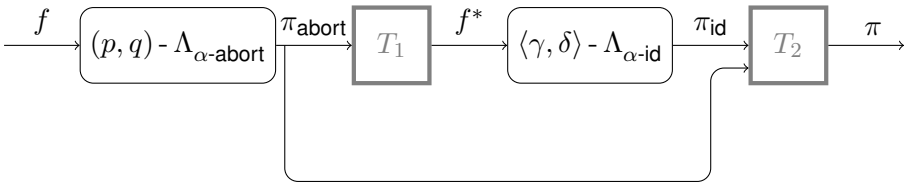

Figure 3: Black-Box Transformation from $\{(p, q)\text{-}\Lambda_{\alpha\text{-abort}}, p'\text{-}\Lambda_{\alpha\text{-id}}\}$ to $\langle\gamma, \delta\rangle\text{-}\Lambda_{\alpha\text{-id}}$, where $\gamma = p$ and $\delta = p' \cdot (p + q)$.

The protocol scheme claimed in Theorem 7 is shown in Figure 3. The first node is a protocol node of $p\text{-}\Lambda_{\alpha\text{-abort}}$, which converts a functionality f into a protocol π_{abort}.

The second node is a black-box node T_1, which converts the protocol π_{abort} to an (n-party) functionality f^*, in which the trusted party takes the view of each party in an execution of π_{abort} as the input, carries out the execution of π_{abort}, and identifies a set of two parties which have inconsistent views, if it exists.[11] When there is none, it outputs $\emptyset$. The third node $\Lambda_{\alpha\text{-id}}$ compiles f^* into a protocol π_{id}.

Finally, a black-box node T_2 combines π_{abort} and π_{id} together and transforms them into a protocol π, which works as follows: initially the parties execute π_{abort} on the given input, and on finishing this execution successfully, each party broadcasts "done." If all parties broadcast "done," then each party outputs the output from the execution of π_{abort} and terminates. If not, they execute π_{id} with their views in the execution of π_{abort} as input. If this latter execution itself aborts, π_{id} identifies a set of parties S at least an α fraction of which is corrupt (where $\alpha \leq 1/2$). otherwise (i.e., if π_{id} finishes without an abort event), then all parties agree on the output of f^*, namely a set S of two parties at least one of which is corrupt, or the emptyset $\emptyset$; if the output is $\emptyset$, the parties set S to be the singleton set consisting of the lexicographically smallest party who did not broadcast "done" after the execution of π_{abort}. In all cases, if π_{abort} resulted in an abort, the honest parties agree on a set of parties S of which at least an α fraction is corrupt.

We verify that the complexity of π is as claimed in the theorem. When there is no abort event, the communication cost is essentially the same as that of π_{abort}, namely $p(n, k)$; otherwise, there is an additional the cost from π_{id}, which is $\widetilde{O}(p(n, k) + p'(n, k) \cdot \mathsf{size}(f^*))$, where $\mathsf{size}(f^*) = \widetilde{O}((p(n, k) + q(n, k)) \cdot \mathsf{size}(f))$. Hence the whole scheme is in $\langle\gamma, \delta\rangle\text{-}\Lambda_{\alpha\text{-id}}$ with $\gamma = p$ and $\delta' = p' \cdot (p + q)$.

Combining Theorem 7 with Theorem 4 we get the following result. Here we state it as efficiency leveraging for full-security; however, the result holds as a BBT from $\{\Lambda_{\mathsf{layer}[d]}, (p, q)\text{-}\Lambda_{\alpha\text{-abort}}, p'\text{-}\Lambda_{\alpha\text{-id}}\}$ as well.

Theorem 8. *For all $0 \leq \alpha \leq 1/2$, and for all functions $p, q, p' \in \mathrm{poly}(n, k)$, there exists a BBT from $\{\Lambda_{\mathsf{layer}[d]}, (p, q)\text{-}\Lambda_{\alpha\text{-abort}}, p'\text{-}\Lambda_{\alpha\text{-full}}\}$ to $(p; \mathsf{D})\text{-}\Lambda_{\alpha\text{-full}}$, where $d = \frac{n \cdot p' \cdot (p+q)}{p}$ and $\mathsf{D}(f) = \mathsf{width}_d(f)$.*

9 Applications

In Section 4.1, we already saw a pedagogical application of BBT, in simplifying the exposition of security with guaranteed output delivery (with computationally bounded adversaries). In this section, we give several

[11] Recall that the view of a party involves its initial input, the randomness, and all the received messages.

interesting examples regarding how to use the BBTs in the previous sections for deriving both feasibility and efficiency results.

∘ **Rabin-Ben Or without honest-majority VSS.** As our first example, we reproduce the classic feasibility result of Rabin and Ben-Or [28] for fully secure MPC for corruption against $t < n/2$ parties. The core new tool developed in this paper (and used in subsequent results in this regime of corruption) was Verifiable Secret-Sharing (VSS) that is secure against corruption of $t < n/2$ parties. Interestingly, our construction by-passes the need for an explicit VSS protocol for this corruption regime, instead showing that one can directly use fully secure MPC from prior work [2, 7]. Our construction is based on the following direct corollary of Theorem 5 and Theorem 3.

Corollary 9. *For any $0 < \alpha, \beta \leq 1/2$, there exists a BBT from $\{\Lambda_{\alpha\text{-sh}}, \Lambda_{\beta\text{-full}}\}$ to $\Lambda_{\alpha\text{-full/BC}}$.*

To obtain the result of [28] we simply apply Corollary 9 to the protocols in [2, 7].

∘ **Constant-Rate MPC with Full-Security for Small Number of Parties.** Our first quantitative result is a "constant-rate" honest-majority MPC protocol with guaranteed output delivery, *when the number of parties involved is constant*. That is, as the size of the function grows, the communication complexity of the protocol grows linearly at a rate that is independent of the security parameter. For MPC of large circuits, against the optimal corruption threshold $n/2$, this gives an amortized complexity of $O(1)$ per gate, compared to $O(k)$ per gate in the previously best result from [3].

Corollary 10. *There exists a p-$\Lambda_{1/2\text{-full/BC}}$-scheme, where $p(n,k) = c_n$ is independent of k.*

This result is obtained as a corollary of Theorem 6[12] and Theorem 3. First we obtain a p-$\Lambda_{1/2\text{-id/BC}}$ scheme by applying the BBT from Theorem 6 to the $\Lambda_{1/2\text{-sh}}$-scheme from [2] and the constant rate $\Lambda_{\beta\text{-full}}$-scheme (for some $\beta > 0$) that is obtained by instantiating the protocol scheme from [13] using the constant-rate ramp scheme of [8]. (The same "outer protocol" was used in [23] to obtain a constant-rate $\Lambda_{\text{abort/OT}}$-scheme.) Then by further applying the BBT fromTheorem 3, we obtain the p-$\Lambda_{1/2\text{-full/BC}}$ protocol as claimed.

∘ **Scalable MPC with Full-Security, Optimal Threshold.** Our next result is a "scalable" honest-majority MPC protocol with guaranteed output delivery. We define the function class $\mathcal{F}_{\text{arith}}$ of functions represented as arithmetic circuits over a field $\mathbb{F}$ such that $\log|\mathbb{F}| > k$. For $f \in \mathcal{F}_{\text{arith}}$, $\mathsf{size}(f)$ refers to $\log|\mathbb{F}| \cdot |C_f|$, where $|C_f|$ is the number of gates in the circuit C_f representing f. Equivalently, $\mathsf{size}(f)$ measures the number of binary wires in the circuit C_f; similarly $\mathsf{width}(f)$ measures the width of C_f in bits.

Corollary 11. *There exists a $(p; \mathsf{D})$-$\Lambda^{\mathcal{F}_{\text{arith}}}_{\frac{1}{2}\text{-full/BC}}$-scheme, where $p(n,k) = n$ and $\mathsf{D} = \mathsf{width}(f)$.*

That is, for MPC of large arithmetic circuits over a large field, with security against the optimal corruption threshold $n/2$, we get an amortized communication cost of $O(n)$ bits per binary wire in the circuit. This result is obtained as a corollary of Theorem 7 and Theorem 4, by applying the BBTs to the $\Lambda^{\mathcal{F}_{\text{arith}}}_{1/2\text{-abort}}$-scheme from [15] and the p-$\Lambda_{1/2\text{-id}}$-scheme from Corollary 10. Note that we have used $\mathsf{width}(f)$ as an upper-bound on $\mathsf{width}_d(f)$ over all d.

[12] The construction leading to Theorem 5 also suffices here. We point to Theorem 6 only because it makes the parameters explicit; the optimization in Section 7.1 is not important for this result.

Our result complements a similar result of Ben-Sasson et al. [3] in which the secondary complexity measure is depth, instead of width. We remark that a natural regime for scalable MPC involves long sequential computations (carried out by a small or moderate number of parties), so that a circuit for the computation would be deep and narrow. In such a regime, the above result, which yields a cost of $O(n \cdot \mathsf{size}(f) + \mathrm{poly}(n,k))$, compares favorably to the protocols of [3] which yield a cost of $\widetilde{\Omega}(n \cdot \mathsf{size}(f) + n^2 \cdot \mathsf{depth}(f) + \mathrm{poly}(n,k))$.

∘ **Highly Scalable MPC with Full-Security, Near Optimal Threshold.** Our final application considers the problem of relaxing the corruption threshold from the optimal $\alpha = 1/2$ to $\alpha = 1/2 - \epsilon$, for any constant ϵ.

Corollary 12. *For every $\epsilon > 0$, there exists a $(p_\epsilon; \mathsf{D})$ - $\Lambda_{(\frac{1}{2}-\epsilon)\text{-full/BC}}$-scheme, where $p_\epsilon(n,k) = c_\epsilon$ is independent of n and k and $\mathsf{D}(f) = \mathsf{depth}(f)$.*

This generalizes a result in [14], which obtained a similar result (without using a broadcast channel) for the threshold $\frac{1}{3} - \epsilon$. We obtain this result by applying Proposition 1 (Bracha's efficiency leveraging transformation) to our c_n - $\Lambda_{\frac{1}{2}\text{-full/BC}}$ scheme from Corollary 10 and the $(c_1, c_2; \mathsf{depth})$ - $\Lambda_{\beta\text{-full}}$ scheme from [14] (for, say, $\beta = 1/6$ and c_1, c_2 being constants), with $\alpha = 1/2$.

References

[1] M. Ben-Or and R. Cleve. Computing algebraic formulas using a constant number of registers. *SIAM Journal on Computing*, 21(1):54–58, 1992.

[2] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. 20th STOC*, pages 1–10. ACM, 1988.

[3] E. Ben-Sasson, S. Fehr, and R. Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. In *Proc. 32th CRYPTO*, pages 663–680. Springer, 2012.

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

[5] S. Cabello, C. Padró, and G. Sáez. Secret sharing schemes with detection of cheaters for a general access structure. *Des. Codes Cryptography*, 25(2):175–188, 2002.

[6] A. Cevallos, S. Fehr, R. Ostrovsky, and Y. Rabani. Unconditionally-secure robust secret sharing with compact shares. In *Proc. 31th EUROCRYPT*, pages 195–208. Springer, 2012.

[7] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *Proc. 20th STOC*, pages 11–19. ACM, 1988.

[8] H. Chen and R. Cramer. Algebraic geometric secret sharing schemes and secure multi-party computations over small fields. In *CRYPTO*, pages 521–536. Springer, 2006.

[9] R. Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *STOC*, pages 364–369. ACM, 1986.

[10] R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient multiparty computations secure against an adaptive adversary. In *EUROCRYPT '99*, pages 311–326, 1999.

[11] R. Cramer, I. Damgård, and S. Fehr. On the cost of reconstructing a secret, or VSS with optimal reconstruction phase. In *Proc. 21th CRYPTO*, pages 503–523. Springer, 2001.

[12] R. Cramer, Y. Dodis, S. Fehr, C. Padró, and D. Wichs. Detection of algebraic manipulation with applications to robust secret sharing and fuzzy extractors. In *Proc. 27th EUROCRYPT*, pages 471–488. Springer, 2008.

[13] I. Damgård and Y. Ishai. Scalable secure multiparty computation. In *Proc. 26th CRYPTO*, pages 501–520. Springer, 2006.

[14] I. Damgård, Y. Ishai, and M. Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In *Proc. 29th EUROCRYPT*, pages 445–465. Springer, 2010.

[15] D. Genkin, Y. Ishai, M. M. Prabhakaran, A. Sahai, and E. Tromer. Circuits resilient to additive attacks with applications to secure multiparty computation. In *The Proceedings of the 46th Annual Symposium on the Theory of Computing (STOC)*, 2014.

[16] O. Goldreich. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, 2004.

[17] O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. In ACM, editor, *Proc. 19th STOC*, pages 218–229. ACM, 1987. See [16, Chap. 7] for more details.

[18] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proc. 17th STOC*, pages 291–304. ACM, 1985.

[19] C. Hazay and M. Venkitasubramaniam. On the power of secure two-party computation. Cryptology ePrint Archive, Report 2016/074, 2016. http://eprint.iacr.org/2016/074.

[20] M. Hirt and J. B. Nielsen. Upper bounds on the communication complexity of optimally resilient cryptographic multiparty computation. In *Advances in Cryptology - ASIACRYPT 2005, 11th International Conference on the Theory and Application of Cryptology and Information Security, Chennai, India, December 4-8, 2005, Proceedings*, pages 79–99, 2005.

[21] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. In *STOC*, pages 21–30. ACM, 2007.

[22] Y. Ishai, R. Ostrovsky, and V. Zikas. Secure multi-party computation with identifiable abort. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, pages 369–386, 2014.

[23] Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO*, pages 572–591. Springer, 2008.

[24] Y. Ishai, M. Prabhakaran, and A. Sahai. Secure arithmetic computation with no honest majority. In O. Reingold, editor, *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 294–314. Springer, 2009.

[25] M. Jawurek, F. Kerschbaum, and C. Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 955–966, 2013.

[26] J. Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31. ACM, 1988.

[27] J. Perry, D. Gupta, J. Feigenbaum, and R. N. Wright. Systematizing secure computation for research and decision support. In *Proc. 9th SCN*, pages 380–397. Springer, 2014.

[28] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proc. 21st STOC*, pages 73–85. ACM, 1989.

[29] M. Rosulek. Must you know the code of f to securely compute f? In *Proc. 32th CRYPTO*. Springer, 2012.

[30] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11), Nov. 1979.

[31] A. Shamir, R. L. Rivest, and L. M. Adleman. Mental poker. Technical Report LCS/TR-125, Massachusetts Institute of Technology, April 1979.

[32] S. P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 7(1–3):1–336, 2011.

[33] A. C. Yao. Protocols for secure computation. In *Proc. 23rd FOCS*, pages 160–164. IEEE, 1982.

[34] A. C. Yao. How to generate and exchange secrets. In *Proc. 27th FOCS*, pages 162–167. IEEE, 1986.

A Preliminaries

Protocols. We consider a protocol to be defined by a single program: the role of the party will be part of its input. More formally, we define a state space Σ and a message space M of a protocol, with the following conventions. Each state $\sigma \in \Sigma$ includes information about the party (i.e., a serial number), a security parameter, and arbitrary other information. Each element $\mu \in M$ is a set of individual "messages;" a message includes information about the sender (if an incoming message) or the intended receiver (if an outgoing message). If the sender of an incoming message is a special party called the environment (say, serial number 0), then it is an "input" to the protocol, and an outgoing message with the receiver being environment is an "output" from the protocol.

Definition A.1 (Protocol)**.** *A protocol π is a probabilistic Turing Machine that maps a pair in $(\sigma, \mu) \in \Sigma \times M$ to a pair $(\sigma', \mu') \in \Sigma \times M$, such that σ' has the same party-ID and security parameter as σ. Π denotes the set of all protocols.*

By itself, a protocol does not define a multi-party process. Interpreting a protocol as a process involving a certain number of parties (typically specified as part of the initial state of each honest party), a communication network connecting them, an adversary and an environment is left to be part of the security definition (see below).

Also, typically, we shall require a protocol to have additional structure. For example, we typically require that the protocol be efficient: i.e., there is a polynomial p such that on any input (σ, μ), π terminates in $p(k)$ time steps, where k is the security parameter recorded in σ; further, when executed as a multi-party process, the protocol as a whole should produce a "halting" state for each honest party within a polynomial

number of rounds. However, we do not make these requirements part of the definition of a protocol (but again, leave it to the individual security definitions to make any such requirements).

We shall follow the convention that various parameters of a protocol (e.g., an upperbound on the running time) can be learnt from the protocol in a black-box manner (e.g., it outputs the parameters on a special input).

Functionality. Following the "real/ideal" paradigm, the intended functionality of a protocol can be quite generally defined in terms of another protocol. Indeed, an n-party functionality is simply a protocol involving $n+1$ parties, in which the additional party plays the role of a special (trusted) party (and the other n parties are dummies that behave as routers copying messages back and forth between the environment and the special party).

For conceptual clarity, we shall differentiate between "real" protocols (not involving trusted parties) – which we refer to simply as protocols – and functionalities. We shall often refer to a *functionality family* $\mathcal{F}$, which is simply a set of functionalities, i.e., $\mathcal{F} \subseteq \Pi$. We denote the family of all probabilistic polynomial time computable secure function evaluation functionalities by $\mathcal{F}^*$.

The protocols in a typical functionality family consist of several dummy parties and a (trusted) party carrying out the functionality. For example, the functionality of the secure evaluation of a function $f(\mathbf{x_1}, ..., \mathbf{x_n})$ is a protocol in the ideal world, where the parties P_1,..., P_n copy and send their inputs $\mathbf{x_1}, ..., \mathbf{x_n}$ from the environment to P_0 (the trusted party), and receive and pass the outputs of f from P_0 to the environment. We call such a functionality an n-party functionality (though, as a protocol, there are $n+1$ parties involved).

Parametrized Protocols. A parametrized protocol (or functionality) is used to succinctly represent a family of different protocols (or functionalities). Formally, a parameter is simply a common input to the protocol from all the honest parties. One typical parameter to a protocol is the number of parties. Parameters may also include partial specification of the function being evaluated in a secure function evaluation functionality.

We write $\pi[n]$ to denote a protocol π instantiated with a parameter n.

Normal Form Functionality. A normal form functionality f involves two phases for the trusted party. In the first phase (computation phase), the trusted party arbitrarily interacts with the parties, but at every round maintains a vector of "outputs" $(y_1, \cdots, y_n)$ (n being the number of parties other than the trusted party). This output vector can change from round to round. When the first phase terminates,[13] the trusted party enters the second phase (output delivery phase) and delivers y_i to party $\mathcal{P}_i$, for each i.

As an example, in a secure function evaluation (SFE) functionality, in the first round, the trusted party accepts x_i from $\mathcal{P}_i$ for each i, replacing any missing inputs with a default value sets the output $(y_1, ..., y_n) = f(x_1, ..., x_n)$, and leaves the computation phase.

Functionality $f^{\langle r \rangle}$. For any function $r : \mathbb{Z}^+ \times \mathbb{Z}^+ \to \mathbb{Z}^+$, and any normal form functionality f, we define a functionality $f^{\langle r \rangle}$ as follows. All parties in $f^{\langle r \rangle}$ behave exactly as f during the computation phase. In particular, at the end of every round, the trusted party has a well-defined vector of "outputs" for all the other parties. However, the trusted party does not enter the output delivery phase at the end of the computation phase. Instead, on round $r(|f|, k)$, where $|f|$ is a size parameter of f (typically, its input size) and k is the security parameter, it delivers the *current* outputs to the respective parties (along with any other message in that round). This is carried out irrespective of whether the computation phase has terminated or not.

[13] It is not important for the computation phase to ever terminate. This is because we will typically be interested not in a normal form functionality f itself, but in a round-restricted version $f^{\langle r \rangle}$ as defined below.

Security Definitions. Technically, a security definition for a functionality family $\mathcal{F}$ is formalized as a relation $\widehat{\Lambda} \subseteq \mathcal{F} \times \Pi$. The intention is that $(f, \pi) \in \widehat{\Lambda}$ iff π is a secure protocol for f. For a security notion named secure, the corresponding relation will typically be written as $\widehat{\Lambda}_{\mathsf{secure}}$.

We use a *synchronous model* of communication (with rushing adversaries), so that all parties in a protocol proceed in a round-by-round fashion. Note that this is applicable to ideal functionalities too. However, typically we are not interested in the exact number of rounds in the ideal functionality, as long as it finishes within a polynomial number of rounds. Formally, we define $f^{\langle r \rangle}$ as an $r(|f|, k)$-round versionAlso, let the functionality class $\mathcal{F}^+ = \{f^{\langle r \rangle} | f \in \mathcal{F}$ and r polynomial $\}$. Then, for a security notion secure, we define

$$\Lambda^{\mathcal{F}}_{\mathsf{secure}} = \{(f, \pi) \mid (f^{\langle r \rangle}, \pi) \in \widehat{\Lambda}^{\mathcal{F}^+}_{\mathsf{secure}} \text{ for some polynomial } r\}.$$

In this case, we shall say π is a secure-protocol for f.

In Table 2 we name some of the main security definitions considered in our results. For instance, $\Lambda^{\mathcal{F}}_{\alpha\text{-full}}$ includes all pairs (f, π) such that f is a functionality in the family $\mathcal{F}$, and π is a UC-secure protocol with guaranteed output delivery (within a polynomial number of rounds), against computationally unbounded adversaries who may adaptively corrupt strictly less than α fraction of the parties.

When the functionality class is $\mathcal{F}^*$, the family of all polynomial time computable secure function evaluation functionalities, we write $\Lambda^{\mathcal{F}^*}_{\mathsf{secure}}$ as simply $\Lambda_{\mathsf{secure}}$.

We follow the convention that by default a functionality guarantees output delivery, and aborting behavior should be explicitly specified as part of the functionality. Towards this, given f in normal form, we define another normal form functionality $f^{\langle \mathsf{abort} \rangle}$ in which the trusted party delivers outputs to honest parties only if the adversary explicitly asks it do so. (Formally, in $f^{\langle \mathsf{abort} \rangle}$, during the computation phase, the trusted party sends the corrupt parties' outputs to the adversary, and may overwrite the output of any subset of the honest parties by $\perp$ as directed by the adversary.) Then, we define the set $\Lambda_{\alpha\text{-abort}}$ as

$$\Lambda_{\alpha\text{-abort}} = \{(f, \pi) \mid (f^{\langle \mathsf{abort} \rangle}, \pi) \in \Lambda_{\alpha\text{-full}}\}.$$

Also, as an intermediate security notion we shall define $\Lambda_{\alpha\text{-id}}$ which guarantees that on abortion, corruption can be somewhat localized (within two parties, one of whom is guaranteed to be corrupt).

All the security definitions require the protocol to be efficient, and non-erasing (i.e., the new state σ' produced by the protocol retains all the information from the given state σ). In Table 2, we summarize the various notions referred to in this paper.

As mentioned above, the exact specification of the (real and ideal) interactive processes, as well as the conditions on the protocol under which it is considered to be secure are part of these definitions.

B Examples of Black-Box Transformations

GMW compiler The GMW compiler can be illustrated as a BBT circuit as follows.

The first node is a protocol node of a $(\Lambda^{\mathcal{F}}_{\mathsf{sh\text{-}ppt}})$-scheme, which compiles an input functionality $f \in \mathcal{F}$ into a sh-ppt-secure protocol π for f. The second node is a black-box generator, which converts the protocol π to a "zero-knowledge functionality" $f^{R_\pi}_{ZK}$ for a relationship R_π. Here, R_π is defined in terms of the transcripts in a modified protocol π^*, which carries out a coin-tossing-in-the-well step to generate private random tapes for each party, and then runs π using those random tapes. R_π holds between a partial transcript τ of π^* and the next message m if there exists an initial state s_0 such that by running π^* on the given transcripts will result in the next message to be m.

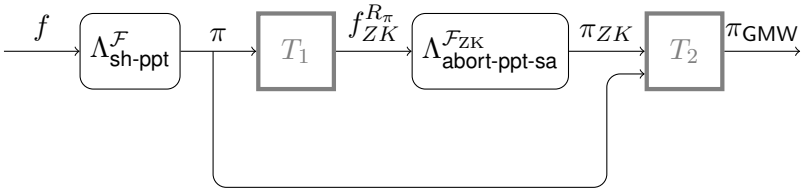

Figure 4: BBT from $\{\Lambda_{\text{sh-ppt}}, \Lambda^{\mathcal{F}_{\text{ZK}}}_{\text{abort-ppt-sa}}\}$ to $\Lambda_{\text{abort-ppt-sa}}$ in the GMW compiler

Next, the third node is a protocol node of $\Lambda^{\mathcal{F}_{\text{ZK}}}_{\text{abort-ppt-sa}}$-scheme which compiles the input functionality $f^{R_\pi}_{ZK} \in \mathcal{F}_{\text{ZK}}$ into a ZKP protocol π_{ZK}. Note that again, π_{ZK} is an abort-ppt-sa-secure protocol for $f^{R_\pi}_{ZK}$. Finally, a black-box node uses π and π_{ZK} to define a protocol π_{GMW}, an abort-ppt-sa protocol for $\mathcal{F}$.

The above transformation is against computationally bounded adversaries, and it uses a one-way function. It is possible to reformulate this result as an unconditional transformation that yields a protocol in the commitment hybrid model. For this, we rely on the IKOS compiler (see below).

Bracha's Transformation. Bracha's compiler [4], originally proposed in the context of Byzantine agreement, and later generalized to MPC protocols (see, e.g., [14]), is a transformation from $\{\Lambda_{\alpha\text{-full}}, \Lambda_{\beta\text{-full}}\}$ to $\Lambda_{(1-\epsilon)\alpha\text{-full}}$, with the efficiency guarantees of the $\Lambda_{\beta\text{-full}}$-scheme translating to the efficiency of the resulting protocol. (A more precise statement, using additional notation, appears in Proposition 1.)

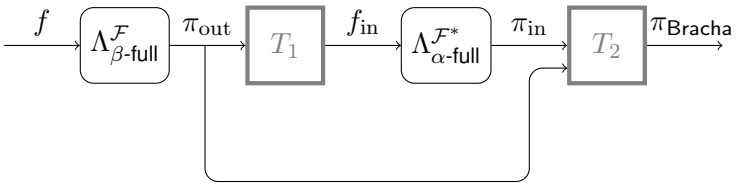

Figure 5: Black-Box Transformation in the Bracha compiler

The first node is a protocol node of a $\Lambda^{\mathcal{F}}_{\beta\text{-full}}$-scheme, which compiles an input functionality f in $\mathcal{F}$ into a so-called outer protocol π_{out}, which is an N-party β-full-secure protocol for f, where α is a small constant, and $N = \text{poly}(n, k)$ depends on the number of parties n of f, the security parameter k and ϵ. Typically, this node will be instantiated with a highly scalable protocol scheme, but with a possibly low threshold β.

The second node is a black-box node, which converts the protocol π_{out} to an d-party functionality (for a small d, typically $d = O(\frac{1}{\epsilon^2})$), in which the trusted party carries out the execution of a party in π_{out} (with all incoming and outgoing messages to it secret-shared among the d other parties in the functionality, using, say, an ECSS scheme). The third node transforms this into a secure protocol π_{in} for this functionality (where, typically this node is instantiated with a protocol scheme that may not well with the number of parties n', but has the desired threshold α).

Finally, a black-box node combines π_{in} and π_{out} together and transforms them into a protocol π_{Bracha}, which is a $(\alpha - \epsilon)$-full secure protocol for f.

IPS compiler [23] The IPS compiler shares a similar structure with the Bracha compiler. Figure 1 depicts this transformation. In this figure, T_0^{IPS} converts an n-party functionality into a functionality involving n

"clients" and N "servers" (originally $N = O(n^2k)$ servers in the setting of [23]). T_1^{IPS} defines f_{in} to be an n-party functionality in which the trusted party carries out the program of a server in the protocol π_{out}. (We remark that for some efficiency results, it is desirable to depend on the server's program having further structure – in particular, that parts of its computation are "known" to different clients. In this case, f_{in} carries out only the other parts of the computation. We refer the reader to the discussion on "Type I" and "Type II" computations in [23].)

Finally, the bulk of the compiler is part of the transformation T_2^{IPS}. In Figure 2 (above), we show the structure of the compiler. IPS_{core} combines π_{out} and π_{in} in such a way that the parties play the role of the clients in π_{out} and each server in it is executed using the protocol π_{in}. In addition, at each round, each party P_j is required to send its view in the ℓ^{th} session of π_{in} to each other party P_i through a watchlist channel $W_\ell^{i \to j}$. Each honest P_j can read the messages on watchlist channels for only a small fraction (typically, k/N) of the inner-protocol sessions. The channel is implemented using one-time pads, which are distributed using a watchlist setup functionality $\mathcal{W}$ (described later in the appendix), which is in turn implemented using a protocol w_{IPS}/OT.

The execution is aborted if any party detects an inconsistency among all the views reported on the watchlist for that server (including its own view) and the protocol π_{in}.

[23] show that for appropriate choice of parameters, the resulting protocol scheme is a $\Lambda_{\text{abort/OT}}$-scheme.

IKOS compiler [21]. The IKOS compiler transforms honest-majority MPC protocols to zero-knowledge protocols. It can be illustrated as a BBT circuit, as follows:

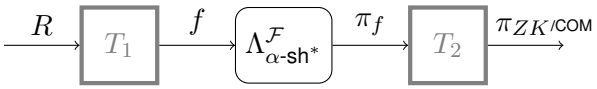

Figure 6: A schematic diagram of the IKOS transformation.

First, T_1 transforms an NP-relation $R(x, w)$ into an n-party functionality $f(x, w_1, ..., w_2) = R(x, w_1 \oplus ... \oplus w_n)$, where $\oplus$ denotes bitwise exclusive-or of strings. Second, a protocol node of $\Lambda^{\mathcal{F}}_{\alpha\text{-sh}^*}$-scheme compiles f into a semi-honest and *perfectly correct protocol* π_f.[14] Then, T_2 converts π_f into a protocol in the commitment hybrid model, in which the prover commits to the views of all parties in an execution of π_f (that it "runs in its head"), and the verifier requests it to open a pair of views and verifies their consistency.

To achieve a positive soundness, the above version depends on π_f being at least a 2-private protocol. An alternate version of the IKOS compiler obtains positive soundness with a 1-private protocol π_f: here the prover commits to the views of all parties, *as well as to the communication on each edge*; the verifier picks a single party and gets to see its view as well as the communication on all edges incident on it, and again verifies consistency. This is the version used in the transformation below.

Improving Over [19]. As mentioned in Section 4, we present a simple BBT that improves over a transformation presented in Hazay and Venkitasubramaniam [19]. Our transformation may start with a two-party protocol in the OLE-hybrid model (see Footnote 9) and it uses only a constant number of commitments (to long strings).

[14] As stated in [21], to improve the efficiency of the whole protocol, this protocol node can be replaced by an actively secure protocol with a (lower) constant threshold.

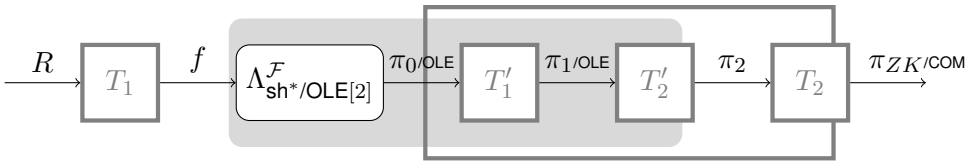

Figure 7: A schematic diagram of the new transformation that improves on a result of [19]. T_1 and T_2 are identical to those in Figure 6. The shaded components constitue a BBT from 2-party, perfectly correct, semi-honest MPC in the OLE-hybrid to a 3-party, perfectly correct, 1-private MPC in the plain model.

Our transformation, shown in Figure 7, follows the IKOS-compiler closely (compare with Figure 6 above). The difference is in the introduction of two intermediate transformations (T'_1 and T'_2) to the (semi-honest) 2-party protocol π_0 for computing f in the OLE-hybrid model, before applying the transformation T_2 from above.

1. Transformation T'_1: Convert π_0 into a 2-party protocol π_1 in the OLE-hybrid model, where all OLE calls are made on random inputs at the beginning of the protocol. Then, whenever π_0 needs to use an OLE-call with inputs $((a,b),x)$ (with the two parties one being the sender and the other being the receiver), we use the following (perfectly secure) reduction of OLE to random-OLE. Given a random-OLE instance between the sender and receiver with inputs $((r_a,r_b),r_x)$, where the receiver got $v = r_a r_x + r_b$, they do the following: the receiver first sends $\Delta = x - r_x$ (since r_x is random this gives no information about x). Then, the sender replies with two messages: $m_1 = a - r_a$ (which, even conditioned on v, r_x and a, b, x, is random) and $m_2 = \Delta r_a + (b - r_b)$ (which, as we shall see, can be solved from $ax + b$ and the other values above). Hence (m_1, m_2) (along with v, r_x) can be perfectly simulated based on $x, ax + b$. Finally the receiver computes $m_1 x + m_2 + v = (a - r_a)x + (x - r_x)r_a + (b - r_b) + r_a r_x + r_b = ax + b$, as needed. (This also shows that m_2 can be derived from m_1, x, v and $ax + b$.)

2. Transformation T'_2: Convert π_1 into a 1-private 3-party protocol π_2 over secure point-to-point channels by just using the third party to implement the OLEs (namely, whenever in π_2 parties P_1, P_2 invoke the OLE with inputs $((r_a,r_b),r_x)$, they now send their inputs to P_3 who evaluates the outcome of this OLE and sends it to the receiver). Note that the views of P_1, P_2 are identical to what their views were in π_2 and that the messages sent to P_3 are completely random and independent of the actual inputs of the OLE. Hence, π_2 is also 1-private.

3. Finally, apply step T_2 of the (1-private variant of the) IKOS transformation to π_2. This gives, a ZK protocol with six string commitments (a view for each of the 3 parties and the communication transcript for each of the 3 channels).

We remark that the above transformation could be seen as a composition of two BBTs. The first two of the three steps above describe a simple BBT from a 2-party semi-honest MPC protocol scheme in the OLE-hybrid to a 3-party 1-private MPC protocol scheme in the plain model (shown shaded in Figure 7). Replacing the 1-private semi-honest protocol in the IKOS BBT of Figure 6 with this transformation yields the full BBT in Figure 7.

C Proofs of Results in Section 5

C.1 Proof of Theorem 1

Firstly, note that the second part of the theorem statement follows from the first part, since a BBT from Λ to $\Lambda^{\mathcal{F}^*}_{\mathsf{secure}}$ is also a BBT from Λ to $\Lambda^{\mathcal{F}}_{\mathsf{secure}}$ for any Λ and any $\mathcal{F} \subseteq \mathcal{F}^*$.

To prove the first part, for $\alpha \in \{0,1\}^k$, let the function $f_\alpha : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}$ be defined as $f_\alpha(x,y) = 1$ iff $x \oplus y = \alpha$. Then we define the functionality family $\mathcal{F} = \{f_\alpha | \alpha \in \{0,1\}^k\}$ (where k is the security parameter).

We claim that there is no BBT from $\emptyset$ to $\Lambda^{\mathcal{F}}_{\mathsf{sh}}$. Suppose for the sake of contradiction π is such a transformation. That is, π^{f_α} is a semi-honest secure protocol for f_α, for all $\alpha \in \{0,1\}^k$. Then first we consider the following experiment. Pick x, y, α uniformly and independently at random from $\{0,1\}^k$, and then let two parties P_1 and P_2 execute the protocol π^{f_α} with inputs x and y respectively, and random tapes $r = (r_1, r_2)$ for P_1 and P_2. We define the following two events.

(A) Either party queries their oracle with (p,q) such that $p \oplus q = \alpha$.

(B) Either party queries their oracle with (p,q) such that $p \oplus q = x \oplus y$.

It is not hard to see that the probability of both these events should be negligible. The probability of event A is negligible since α is chosen uniformly at random, and the parties make only a polynomial number of queries. The reason for the probability of event B being negligible is the security of the protocol. Suppose, w.l.o.g, that the probability that party P_1 makes a query (p,q) such that $p \oplus q = x \oplus y$ is non-negligible. Then, consider an adversary who corrupts P_1 and outputs the list of values $p \oplus q \oplus x$ for every pair (p,q) queried by P_1. By assumption, the probability that this list contains y is non-negligible. However, in the execution of the functionality (ideal world), since $f(x,y) = 0$, an adversary who corrupts only one party has negligible probability of outputting a list of values containing the other party's input (even if it is given α). That is, a simulator (even though it may depend on α) cannot produce such a list. This implies that the adversary in the real execution that we defined above cannot be simulated, contradicting the security of the protocol.

Let R denote the set of all (r, x, y, α) such that in the above experiment, neither event A nor B occurs. Then, as argued above, $\Pr[(r,x,y,\alpha) \in R] > 1 - \mathrm{negl}(k)$.

Now, consider a second coupled experiment in which we use the same r, x, y as in the first experiment, but run $\pi^{f_{\alpha^*}}$, where $\alpha^* = x \oplus y$. We claim that for every $(r,x,y,\alpha) \in R$, the new experiment proceeds identically as the original one. Indeed, it is easily seen using an inductive argument that in both experiments – which differ only in the oracle used – all the oracle queries will be answered by 0. In particular, the output of the protocol, denoted by z, will be identical in both the experiments if $(r,x,y,\alpha) \in R$.

Since $\Pr[(r,x,y,\alpha) \in R] > 1 - \mathrm{negl}(k)$, the distributions of z in the two experiments are at most negligibly apart. However, the correctness of the protocol requires that $\Pr[z = 1] = \mathrm{negl}(k)$ in the first experiment and $\Pr[z = 1] = 1 - \mathrm{negl}(k)$ in the second experiment, leading to a contradiction.

C.2 Proof of Theorem 2

The functionality family $\mathcal{G}$ that we use to prove Theorem 2 corresponds to Zero-Knowledge Proofs. In a functionality $f_R \in \mathcal{G}$, where R is a predicate, the trusted party accepts a pair of inputs (x,w) from P_1 (prover) and sends $(x, R(x,w))$ to P_2 (receiver).

We note that there is a trivial BBT from $\emptyset$ to $\Lambda^{\mathcal{G}}_{\mathsf{sh}}$: consider a protocol in which, on input (x, w), the prover accesses the program of the functionality to compute $R(x, w)$ and sends $(x, R(x, w))$ to the verifier. (Indeed, this is true for all deterministic functionalities in which at most one party has any input.)

So, to prove our theorem, we need to only show that there is no BBT from $\emptyset$ to $\Lambda^{\mathcal{G}}_{\mathsf{abort}}$. This is because, if there were a BBT from $\Lambda^{\mathcal{G}}_{\mathsf{sh}}$ to $\Lambda^{\mathcal{G}}_{\mathsf{abort}}$, then we can replace all its protocol nodes (all labeled with $\Lambda^{\mathcal{G}}_{\mathsf{sh}}$) with the BBT from $\emptyset$ to $\Lambda^{\mathcal{G}}_{\mathsf{sh}}$, and obtain a BBT with no protocol nodes – i.e., a BBT from $\emptyset$ to $\Lambda^{\mathcal{G}}_{\mathsf{abort}}$.

Now, to prove that there is no BBT from $\emptyset$ to $\Lambda^{\mathcal{G}}_{\mathsf{abort}}$, we consider $\mathcal{G}_0 \subseteq \mathcal{G}$ as follows. For the sake of exposure, first we define $\mathcal{G}_0$ as a family of inefficient functionalities, and later use our cryptographic assumption to replace it with an efficient verion.

For every (possibly inefficient) function $O : \{0,1\}^k \to \{0,1\}^{2k}$, let $R_O(\cdot, \cdot)$ be a relationship such that

$$R_O(x, w) = \begin{cases} 1, & \text{if } O(w) = x \\ 0, & \text{otherwise.} \end{cases}$$

For the sake of contradiction, suppose there is a BBT transformation from $\emptyset$ to $\Lambda^{\mathcal{G}}_{\mathsf{abort}}$. That is, let T be an oracle TM such that for all $R_O \in \mathcal{G}_0$, T^{R_O} is a zero-knowledge protocol for the relation R_O. Let (T_1, T_2) denote the programs obtained by specializing T for the prover (P_1) and the verifier (P_2) respectively (so that the protocol consists of the pair of programs $(T_1^{R_O}, T_2^{R_O})$).

We shall argue that if this protocol is complete and zero-knowledge, then it is not sound. To show this, we define a cheating prover $\hat{P}^{R_O}$, which on input x', picks $w' \leftarrow \{0,1\}^k$ and runs $T_1^{Z^{R_O}_{(x',w')}}$, where the oracle TM $Z_{(x',w')}$ is defined as follows:

$$Z^R_{(x',w')}(x, w) = \begin{cases} 1 & \text{if } (x, w) = (x', w') \\ 0 & \text{if } w = w', x \neq x' \\ R(x, w) & \text{otherwise.} \end{cases}$$

We claim that if the original protocol is zero-knowledge, then the probabilities of the verifier accepting in the following two experiments differ by at most a negligible quantity:

E1. Let $O : \{0,1\}^k \to \{0,1\}^{2k}$ be a random function. Sample $w \leftarrow \{0,1\}^k$, and let $x = O(w)$. P_1 runs $T_1^{R_O}$ with input (x, w), against P_2 running $T_2^{R_O}$.

E2. Let $O : \{0,1\}^k \to \{0,1\}^{2k}$ be a random function. Sample $x \leftarrow \{0,1\}^{2k}$. P_1 runs $\hat{P}^{R_O}$ with input x, against P_2 running $T_2^{R_O}$.

To see this, we consider modifying experiment E1 as follows:

H1. Let $O : \{0,1\}^k \to \{0,1\}^{2k}$ be a random function. Sample $w \leftarrow \{0,1\}^k$, and let $x = O(w)$. Further, sample $x' \leftarrow \{0,1\}^{2k}$, and define O' to be the same as O, except $O(w) = x'$. P_1 runs $T_1^{R_O}$ with input (x, w), against P_2 running $T_2^{R_{O'}}$.

H2. Let $O' : \{0,1\}^k \to \{0,1\}^{2k}$ be a random function. Sample $x \leftarrow \{0,1\}^{2k}$. P_1 runs $\hat{P}^{R_{O'}}$ with input x, against P_2 running $T_2^{R_{O'}}$.

Note that we have changed the oracle for P_2 to be $R_{O'}$. But we claim that the outcome of this experiment remains close to the original one, due to the zero-knowledge property. Indeed, $R_{O'}$ and R_O are identical

on all queries of the form (a, b) as long as $b \neq w$. By the zero-knowledge property and the nature of the ideal functionality, in experiment E1, the probability that an (honest-but-curious) adversary corrupting P_2 can output a polynomial sized list containing w is negligible. That is, except with negligible probability, in experiment E1, P_2 never queries its oracle on an input of the form $(\cdot, w)$. Hence it follows (from an inductive argument similar to that in the previous proof) that the outcome of E1 remains indistinguishable from that of H1.

Next, we note that the experiments H1 and H2 are in fact identical. To see this, note that $\hat{P}^{R_{O'}}$ on input x provides T_1 with access to the oracle R_O where O and O' are identical, except that for a randomly sampled w, $O(w) = x$ whereas $O'(w) = x'$ for independently sampled x, x', and (x, w) is the input to T_1.

Finally, H2 and E2 are identical (except for renaming O to O'). This shows that the probabilities of P_2 accepting in E1 and E2 are negligibly different. By completeness, the first probability should be close to 1, and hence so is the second probability. However, the statement being proven in E2 is false (except with negligible probability), because it is extremely unlikely that x falls in the image of O. Hence this violates the soundness requirement.

This concludes the argument that there is no BBT from $\emptyset$ to $\Lambda^{\mathcal{G}_0}_{\mathsf{abort}}$. However, the functionality class $\mathcal{G}_0$ as defined above is not a valid (efficient) functionality, since not all functions $O : \{0,1\}^k \to \{0,1\}^{2k}$ are efficiently realizable. We can easily resolve this by considering the set of functions defined by a pseudorandom function $F : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}^{2k}$ with k-bit seeds, k-bit inputs and $2k$-bit outputs. Then, in the above proof, the experiments E1 and E2 remain indistinguishable if the random function O is replaced by $F(s, \cdot)$ where $s \leftarrow \{0,1\}^k$.

D A Simple Error Correcting Secret-Sharing Scheme

We defined ECSS in Section 2.3. For the sake of completeness, below we include an elementary (n, t)-ECSS scheme for any $t \leq n/2$. A more efficient scheme was given by [6] improving on previous schemes [28, 11].

$\mathrm{ECSS} - \mathrm{Share}[t, n](x)$ (where $t \leq n/2$):

1. $(\rho_1, ...\rho_n) \leftarrow \mathsf{share}[t, n](x)$
2. Pick $n(n-1)$ one-time MAC keys: $\{K_{i,j}\}_{i,j \in [n], i \neq j}$
3. Let $\tau_{i,j} = \mathrm{MAC}_{K_{i,j}}(\rho_j) \forall i, j \in [n], i \neq j$
4. For all i, set $\sigma_i = (\rho_i, \{K_{i,j} | j \neq i\}, \{\tau_{j,i} | j \neq i\})$
5. Output $(\sigma_1, ..., \sigma_n)$

$\mathrm{ECSS} - \mathrm{Reconstruct}[t, n](\sigma_1, ..., \sigma_n)$**:**

1. Define $\mathsf{consistent}(\sigma_i, \sigma_j) =$ True iff $\tau_{i,j} = \mathrm{MAC} K_{i,j}(\rho_j)$ and $\tau_{j,i} = \mathrm{MAC}_{K_{j,i}(\rho_i)}$.
2. Find a subset $S \subseteq [n], |S| > n - t$ such that $\forall (i, j) \in S, \mathsf{consistent}(\sigma_i, \sigma_j)$. If no such S exists, output a default value; otherwise, let $S^* \subseteq S, |S^*| = t$ be the lexical smallest t indices in S and output Reconstruct$\{\rho_i\}_{i \in S^*}$ where ρ is part of σ_i.

E Details Omitted from Section 6

E.1 Details of the Construction in Theorem 3

In this section we present the details behind Theorem 3. The outline of our BBT is illustrated in Figure 8(a).

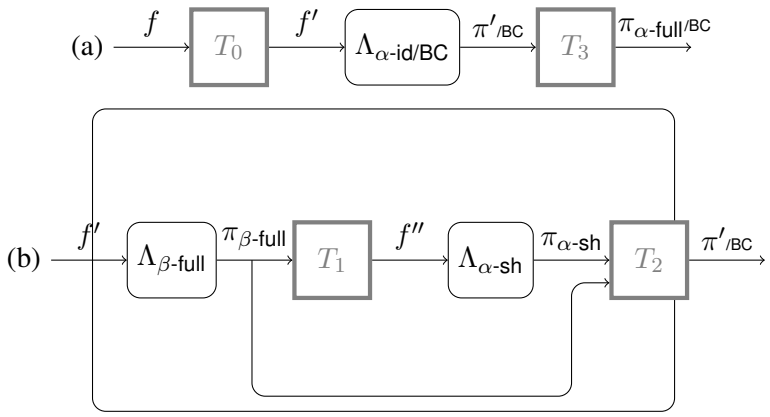

Figure 8: (a) BBT from $\{\Lambda_{\alpha\text{-id/BC}}\}$ to $\Lambda_{\alpha\text{-full/BC}}$. (b) BBT from $\{\Lambda_{\beta\text{-full}}, \Lambda_{\alpha\text{-sh}}\}$ to $\Lambda_{\alpha\text{-id}_{\frac{1}{2}}\text{/BC}}$. (a) and (b) together they yield a BBT from $\{\Lambda_{\beta\text{-full}}, \Lambda_{\alpha\text{-sh}}\}$ to $\Lambda_{\alpha\text{-full/BC}}$ for any $\alpha \leq \frac{1}{2}$.

First, we define the transformation T_0, which essentially wraps f between a layer of reconstruct and a layer of share. More precisely, T_0 transforms f into a parametrized functionality $f' = T_0^f$ which accepts shares of f's inputs and outputs shares of f's output. To define $f'[n']$, where the parameter n' is the number of parties in f', fix an $(n', n'/2)$-ECSS scheme $(\mathsf{share}_{n'}, \mathsf{reconstruct}_{n'})$. Also, let $(\mathsf{share}^n_{n'}, \mathsf{reconstruct}^n_{n'})$ denote n parallel instances of these algorithms, where $\mathsf{share}^n_{n'}$ takes n secrets and produces n' n-dimensional share vectors, and $\mathsf{reconstruct}^n_{n'}$ takes n' such vectors and outputs n secrets. In f', for each $j \in [n']$, the input of the j^{th} party is parsed as $\Sigma_j = (\sigma_j^1, \cdots, \sigma_j^n)$; then

$$f'(\Sigma_1, \cdots, \Sigma_{n'}) = \mathsf{share}^n_{n'}(f(\mathsf{reconstruct}^n_{n'}(\Sigma_1, \cdots, \Sigma_{n'}))). \tag{2}$$

Note that f' is defined using f in a blackbox manner, and hence can indeed be implemented as T_0^f.

Next we describe the transformation T_3 which defines the final protocol in terms of a protocol $\pi_{\alpha\text{-id}/\text{BC}}$ for $f^{\langle \mathsf{id}_\alpha \rangle}$. A formal description appears in Appendix F.1. Conceptually, T_3 is quite simple: it maintains a set of "active parties" $T^* \subseteq [n]$, such that more than $(1-\alpha)$ fraction in this set is honest. Initially, T^* is the set of all parties, and the protocol goes through one or more iterations of the execution of $\pi_{\alpha\text{-id}/\text{BC}}$ among the active parties. Since $\pi_{\alpha\text{-id}/\text{BC}}$ UC-securely realizes the functionality $f'^{\langle \mathsf{id}_\alpha \rangle}$, we can consider these executions of the latter. In each iteration, all parties freshly share their inputs for f to the active parties using an $(n', \alpha n')$-ECSS scheme, where $|T^*| = n'$, which they use as their inputs to $f'^{\langle \mathsf{id}_\alpha \rangle}$. If $f'^{\langle \mathsf{id}_\alpha \rangle}$ aborts, then the adversary does learn the outputs of f'; however, these outputs, being less than αn shares of the f-output of each party, are independent of the inputs of the honest parties to f. Further, since each time an abort happens T^* shrinks by at least one party, this can happen at most $O(n)$ times. Also, note that removing the set of parties identified by $f'^{\langle \mathsf{id}_\alpha \rangle}$ can only increase the fraction of honest parties in T^*, and hence maintains the invariant that less than $\alpha n'$ of the n' active parties are corrupt, as required for the next

iteration. Eventually, an execution of $f'^{\langle \mathsf{id}_\alpha \rangle}$ completes without aborting. Note that in this execution, f' correctly reconstructs the inputs of all honest parties, due to the error-correcting nature of the ECSS scheme. Thus the honest active parties will receive ECSS shares of the correct outputs for all the n parties. They send the shares to the respective parties, who can correctly reconstruct their outputs (since they would have received more than $(1-\alpha)n'$ shares from the honest parties.

E.2 Details of the Construction in Theorem 4

The new transformation, which is similar to that in Appendix E.1 is shown in Figure 9 (compare with Figure 8(a)).

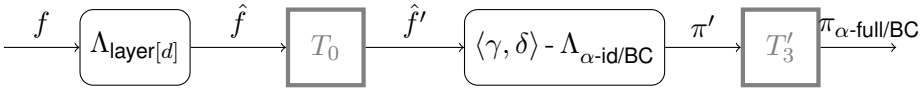

Figure 9: A Streamlined version of the BBT in Figure 8(a), using layered circuits.

The transformation T_0 here is identical to that in Figure 8(a). T_3' is obtained by modifying T_3 so that it carries out the evaluation of one layer at a time, using the outputs from one layer as the inputs to the next. Furthermore the set of active parties are maintained across all the executions.

The security of this BBT follows along the same lines as the one in Appendix E.1. Below we verify the complexity of this transformation. Let $\hat{f}'[i]$ denote the analog of f' in Equation 2. Then, $\mathsf{size}(\hat{f}'[i]) = O(\mathsf{size}(f)/d + e(n,k) \cdot \mathsf{width}_d(f))$, where $e(n,k)$ denotes the computational overhead for implementing ECSS, per bit of input ($e(n,k) = O(n^2 k)$ from Appendix D).

Now, if there is no abort event, by the definition of $\langle \gamma, \delta \rangle$ - $\Lambda_{\alpha\text{-id}}$, the communication cost is $O(\sum_{i=1}^{d} \gamma(n,k) \cdot \mathsf{size}(\hat{f}'[i]) + \mathrm{poly}(n,k))$, which is

$$O(\gamma(n,k) \cdot \mathsf{size}(f) + d(n,k) \cdot \gamma(n,k) \cdot e(n,k) \cdot \mathsf{width}_d(f) + \mathrm{poly}(n,k)).$$

Each time an abort happens at level i, it results in an additional cost of $O(\delta(n,k) \cdot \mathsf{size}(\hat{f}'[i]) + \mathrm{poly}(n,k))$. But, the number of times an abort event can occur is $O(n)$ (since in each abort a non-empty set of players are eliminated from the set of active parties). Thus the additional cost due to all aborts is $O(n \cdot \delta(n,k) \cdot \mathsf{size}(f)/d + n \cdot \delta(n,k) \cdot e(n,k) \cdot \mathsf{width}_d(f) + \mathrm{poly}(n,k))$. Substituting d, this cost matches that of the cost without abort events. Hence, the total cost also matches this expression. Finally, noting that $\gamma(n,k)$, $d(n,k)$, $e(n,k)$ are all $\mathrm{poly}(n,k)$ we get the claimed complexity.

Remark. We note that above, if $d(n,k) \cdot e(n,k) \cdot \mathsf{width}_d(f) \leq \mathsf{size}(f)$, then the communication complexity becomes

$$O(\gamma(n,k) \cdot \mathsf{size}(f) + \mathrm{poly}(n,k)).$$

In this case, the above BBT yields a γ - $\Lambda_{\alpha\text{-full}}$ protocol scheme.

F Details of the Construction in Theorem 5

F.1 Formal Description of T_3

Transformation T_3: Given a protocol $\pi_{\alpha\text{-id}}$ (parametrized by number of parties) and a collection of $(n', \alpha n')$-ECSS

schemes $(\mathsf{share}_{n'}, \mathsf{reconstruct}_{n'})$ for all values of n':

1. Set the set of active parties $T^* = [n]$ and `done = false`
2. While `done` is `false`
   (a) $\forall i \in [n]$, P_i shares its input x_i as $(\sigma^i_{j_1}, \cdots, \sigma^i_{j_{n'}}) \leftarrow \mathsf{share}_{n'}(x_i)$ where $T^* = \{j_1, \cdots, j_{n'}\}$; it sends σ^i_j to P_j, for all $j \in T^*$.
   (b) Parties in T^* run the protocol $\pi_{\alpha\text{-}\mathsf{id}}[n']$, with P_j's input being $(\sigma^1_j, ..., \sigma^n_j)$ (using $\perp$ for any σ^i_j that was not received).
   (c) For each $j \in T^*$, if P_j receives the output $(\mathsf{corrupt}, \ell_1, \ell_2)$ from the above protocol, then it lets $T^* = T^* \setminus \{\ell_1, \ell_2\}$ and broadcasts T^*; else P_j gets output $(\delta^1_j, ..., \delta^n_j)$ and broadcasts "done."
   (d) For each $i \in [n]$, on receiving "done" from $n'/2$ or more parties in T^*, P_i sets `done = true`; else it sets T^* to be what was broadcast by a majority of parties in T^*.
3. For each $i \in [n]$, $j \in T^*$, P_j sends δ^i_j to P_i.
4. For each $i \in [n]$, P_i receives $(\delta^i_1, ..., \delta^j_{n'})$ and outputs $\mathsf{reconstruct}_{n'}(\delta^i_1, ..., \delta^i_{n'})$.

F.2 Watchlist Setup

F.2.1 Implementing the Watchlist Functionality $\mathcal{W}^*$

We need to define a protocol w^* that UC-securely implements $\mathcal{W}^*$ in the honest majority setting (in the broadcast-hybrid model). We do this in a few steps: first, we adapt the protocol $w_{\mathrm{IPS}}/\mathsf{OT}$ for $\mathcal{W}$ from [23] to use a weak form of OT that we denote by $\widetilde{\mathsf{OT}}$ so that $w_{\mathrm{IPS}}/\widetilde{\mathsf{OT}}$ still remains a secure protocol for $\mathcal{W}$. Next we build a protocol $\pi_{\widetilde{\mathsf{OT}}}$ that securely realizes $\widetilde{\mathsf{OT}}$ (in the broadcast-hybrid model) given an honest majority, and then compose it with the above protocol to obtain a protocol w (in the broadcast-hybrid model). Finally, we carry out a simple generic transformation to convert w to w^*. We start by describing this step.

T^*_{id}: from a Protocol for $\mathcal{W}$ to one for $\mathcal{W}^*$. We shall require a "well-behaved protocol" w for $\mathcal{W}$: that is, a party aborts in w only if its view is inconsistent with an honest execution of w. Then, one can generically transform such a protocol to a protocol for $\mathcal{W}^*$ as defined above: simply run w, and if it aborts, all parties must broadcast their views in w (including their inputs). Since w is well-behaved, if a party initiating the abort is honest, then it is guaranteed that there will be an inconsistency in the views revealed. Then all the parties would identify the lexicographically first pair of inconsistent parties (i, j) and output it. If no inconsistency is discovered, the pair (i, j) is taken to be the lexicographically first pair in which i is a party initiating the abort.

Protocol w for $\mathcal{W}$. Given the above, we need only build a well-behaved protocol w for $\mathcal{W}$. As mentioned above, our implementation of w involves composing w_{IPS} with an implementation of $\widetilde{\mathsf{OT}}$. First, we formally define the functionality $\widetilde{\mathsf{OT}}$.

Functionality $\widetilde{\mathsf{OT}}$. An n-party functionality (parametrized by a probability $\gamma \in [0, 1]$ and a positive integer ℓ and a field $\mathbb{F}$), which interacts with only two parties – a sender S, receiver R – and the adversary Adv as follows.

1. Accepts $m \in \mathbb{F}^\ell$. Also, accepts $c \in \{0, 1\}$ from Adv.
2. Sample $b \in \{0, 1\}$ according to $p(b = 1) = \gamma$.

3. If $b = 0$, then send erased to R.

4. If $b = 1$ and $c = 0$, then send m to R, else send abort to R.

In analyzing w, its main difference from w_{IPS} is that, since w is in the $\widetilde{\mathsf{OT}}$-hybrid model, the adversary may choose to try and learn if some of the pads were delivered to the honest party or not. However, for each such pad, the adversary has a probability γ of causing abort. Note that $\mathcal{W}$ does allow the adversary to learn the above information for up to δN servers. To show that the protocol remains secure despite using $\widetilde{\mathsf{OT}}$ instead of OT, we need to argue that the the probability of adversary learning this information about more than δN servers without causing an abort is exponentially small in k. Indeed, this probability is $O(1-\gamma)^{\delta N}$. For an appropriate choice of parameters, this probability is exponentially small in k. Specifically, we choose $\gamma = \Theta(k/N)$ as in [23], and set $\delta = \Theta(1)$.

To complete our construction, we present a protocol $\pi_{\widetilde{\mathsf{OT}}}$ that UC-securely realizes $\widetilde{\mathsf{OT}}$ against an adversary who corrupts strictly less than $n/2$ parties. A formal description of this protocol is given in Appendix F.2.

The protocol uses a simple Oblivious Linear Function Evaluation protocol in which the sender sends out Shamir shares of two vectors $a, b \in \mathbb{F}^L$ to all n parties, and the receiver sends out Shamir shares of a scalar $x \in \mathbb{F}$ (chosen from a small range of d elements), so that each of the n parties can locally compute a share of the vector $ax + b$. The degrees of the Shamir shares are chosen appropriately (say, $\lceil \frac{n-1}{2} \rceil$ for a, $\lfloor \frac{n-1}{2} \rfloor$ for x, and $n-1$ for b), so that corrupting strictly less than $n/2$ parties reveals no information about any of these values, but $ax + b$ can be reconstructed from the n shares locally computed by the n parties. The n parties are expected to send their shares to the receiver so that it can reconstruct the vector $ax + b$; then the sender picks β randomly from the same domain as x is chosen from, and sends $(\beta, a\beta + b + v)$ to the receiver. If $\beta = x$ (which happens with probability $1/d$), the receiver can recover v and otherwise, it receives no information about v.

However, in the form described above, this protocol is not a secure Rabin OT protocol since the adversary can easily alter the value being reconstructed by changing its own shares that it sends to the receiver. But the only way the adversary can alter the vector v is to add an independent vector δ to it. To protect against this, we let v be an encoding of the actual message which can detect additive attacks. Such a code is called an algebraic manipulation detection (AMD) code [12]; elementary constructions of AMD codes exist and will suffice for us.[15] Thus, if the adversary alters the shares in any way, *and if* $\beta = x$, then this will be detected and the protocol will abort. However, if $\beta \neq x$, the protocol does not abort, and the adversary can learn this fact. This matches the security guarantee offered by the functionality $\widetilde{\mathsf{OT}}$. Indeed, it can be shown that this protocol is a UC secure protocol for $\widetilde{\mathsf{OT}}$, against active corruption of less than α fraction of the parties, for any $\alpha \leq 1/2$.

F.2.2 Listing of Functionalities and Protocols for Watchlist Setup

We present a slightly modified version of the watchlist functionality used in the IPS compiler [23]. The modification is the addition of the starred item in the description below, which allows the adversary to learn if any of the honest parties are watching a server in the set S that the adversary has gained (possibly partial) access to. While this was not possible in the OT-based construction in [23], the analysis there allows for this (all servers in the set S are considered to be actively corrupted).

[15] For the sake of completeness, we point out a simple construction AMDlite due to Cabello et al. [5], that predates the definition of AMD codes: each element $m \in \mathbb{F}$, where $\mathbb{F}$ is an exponentially large field, is encoded as a triplet (m, s, ms), where $s \leftarrow \mathbb{F}$ is uniformly randomly chosen; vectors are encoded by encoding each coordinate independently.

IPS Watchlist Setup Functionality $\mathcal{W}$. An n-party functionality $\mathcal{W}[K, d, \delta, N]$ is parametrized by the length of pads (communication cost of the watched protocol) K, $d \in [n]$ such that $\gamma = \frac{1}{d}$ is the expected fraction of pads to be delivered (i.e., fraction of servers to be watched by an honest party), $0 \leq \delta < 1$ being the fraction of server for which the adversary can learn if a pad corresponding to it was delivered to any honest party or not, and the number of servers N, which interacts with the honest parties, the corrupted parties, and the adversary Adv as follows.

1. For each honest party P_i, for $\ell \in [N], j \in [n]$, pick a random K-bit string $\mathsf{Pad}_\ell^{i \to j}$ and give it to P_i.
2. For each honest party P_j, pick random $L^j \subseteq [N], |L^j| = \gamma N$, and send $\{\mathsf{Pad}_\ell^{i \to j} | \ell \in L^j, i \in [n]\{j\}$ to P_j.
3. From each corrupted party P_i, for each $\ell \in [N], j \in [n]$, accept a K-bit string $\mathsf{Pad}_\ell^{i \to j}$.
4. From each corrupted party P_j, accept a set of pairs $S^j \subseteq \{(i, \ell) | i \in [n], \ell \in [N]\}$ such that $\forall i \in [n], S_i^j = \{\ell | (i, \ell) \in S^j\}$ is of size $|S_i^j| \leq 2\gamma N$. Send $\{\mathsf{Pad}_i^{i \to j} | (i, \ell) \in S^j\}$ to P_j for each corrupted P_j.
   - $\star$ Also, accept a set $S \subseteq [n], |S| \leq \delta N$ from the adversary. For each honest party P_i, reveal $S \cap L^i$ to the adversary.
5. At any point Adv can ask $\mathcal{W}$ to send abort to any (honest) party.

Watchlist Setup Functionality with $1/2$-id abort, $\mathcal{W}^*$. With same parameters as $\mathcal{W}$.

1. Run $\mathcal{W}^{\langle \mathsf{id}_{1/2} \rangle}$.
2. If $\mathcal{W}^{\langle \mathsf{id}_{1/2} \rangle}$ outputs $(\mathsf{corrupt}, i, j)$ to the honest parties, then send all the parties' inputs to Adv.

Protocol w^* securely realizing $\mathcal{W}^*$:

1. Run a well-behaved protocol w securely realizing $\mathcal{W}$.
2. If a party gets output abort from w, it broadcasts abort to everyone.
3. On receiving a broadcast of abort, all parties reveal their views (randomness, input, and messages received) in w through the broadcast channel.
4. Each party runs a consistency check on the views broadcast and (the lexicographically first) pair of inconsistent parties (i, j) are identified. Each party outputs $(\mathsf{corrupt}, i, j)$.

Watchlist Protocol w for $\mathcal{W}$, in $\widetilde{\mathsf{OT}}$-hybrid. w is parametrized by K, d, N, where K the length of pads (communication cost of the watched protocol), $d \in [n]$ corresponds to a probability $\gamma = \frac{1}{d}$, which is the expected fraction of pads to be delivered (i.e., fraction of servers to be watched by an honest party), and N is the number of servers.

1. Each party P_j picks random $L^j \subseteq [N], |L^j| = \gamma N$.
2. For each $i \in [n], \ell \in [N], j \in [n]$, P_i picks a random K-bit string $\mathsf{Pad}_\ell^{i \to j}$ and sends it to P_j using $\widetilde{\mathsf{OT}}[K, 2\gamma]$.
3. For each $j \in [n], i \in [n]$, if P_j receives less than γN pads, it aborts; else, it picks γN of them and set $\tilde{L}_i^j \in \{\tilde{\ell} | \mathsf{Pad}_{\tilde{\ell}}^{i \to j}$ received$\}$ s.t. $|\tilde{L}_i^j| = |L^j|$.
4. For each $j \in [n], i \in [n]$, P_j picks a random permutation π_i^j s.t. $\pi_i^j(\tilde{L}_i^j) = L^j$ and sends π_i^j to P_i.
5. Then each party P_j uses $\mathsf{Pad}_{(\pi_i^j)^{-1}(\ell)}^{i \to j}$ for $W_\ell^{i \to j}$.

Protocol $\pi_{\widetilde{\mathsf{OT}}}$ that securely realizes $\widetilde{\mathsf{OT}}$. $\pi_{\widetilde{\mathsf{OT}}}$ is parametrized by p, ℓ, d, n where p is a prime, ℓ is a positive integer such that the message space is $\mathbb{Z}_p^\ell$, d is an integer number which implies a $1/d$ probability for transmission (non-erasure), and n is the number of parties. Ingredients include Shamir secret-sharing and a simple AMD code $\mathsf{AMDlite}$ (see Footnote 15) that encodes vectors in $\mathbb{Z}_p^\ell$ into vectors in $\mathbb{Z}_p^{3\ell}$.

Stage 1: Oblivious Linear Function Evaluation

1. Sender generates random vectors $a, b \in \mathbb{Z}_p^{3\ell}$ and (using Shamir's secret sharing scheme) distributes $[a]_p^{(n-1)/2}$, which are shares of a $(n-1)/2$-degree polynomial and $[b]_p^{(n-1)}$, which are shares of a $(n-1)$-degree polynomial to n-parties (including Sender and Receiver).
2. Receiver generates random $x \in \{0, \cdots, d-1\}$ and (by Shamir's secret sharing scheme) distributes $[x]_p^{(n-1)/2}$, which are shares of $(n-1)$-degree polynomial, to n-parties (including Sender and Receiver).
3. n parties compute $[\phi]_p$, where $\phi = ax + b$ (treating x as a scalar from $\mathbb{Z}_p$), which are shares of a $(n-1)$-degree polynomial, locally and send the shares to Receiver.
4. After collecting n shares, Receiver restores the secret value ϕ as the output. If Receiver cannot collect shares from some parties, it aborts.

Stage 2: Convert to Rabin-OT

1. Sender picks random $\beta \in \{0, \cdots, d-1\}$ and calculates $\hat{m} = m' + a\beta + b$, where $m' = \mathsf{AMDlite}(m)$, and $m \in \mathbb{Z}_p^\ell$ is its input. It sends $(\hat{m}, \beta)$ to Receiver.
2. If $\beta = x$, Receiver applies $\mathsf{AMDlite}$ decoding to $\tilde{m} = \hat{m} - \phi$; if there is no error it outputs the decoded message as the received message; otherwise, Receiver aborts. On the other hand, if $\beta \neq x$, Receiver outputs $\perp$ to indicate erasure.

F.3 Formal Description of T_{id}

The components of the transformation T_2 were illustrated in Figure 2. In this section we formally describe a part of this transformation that adds $1/2$-identifiability, namely T_{id}.

Partial-identification Transformation. From π from the IPS core in the $\mathcal{W}$-hybrid to π^* in the $\mathcal{W}^*$-hybrid and the broadcast channel, where on abort, π^* will output $(\mathsf{corrupt}, i, j)$ such that either P_i or P_j is $\mathsf{corrupt}$:

1. Replace access to $\mathcal{W}$ with access to $\mathcal{W}^*$.
2. If $\mathcal{W}^*$ outputs $(\mathsf{corrupt}, i, j)$ at any round, π^* outputs $(\mathsf{corrupt}, i, j)$.
3. If an abort occurs, let P_i be the lexicographically smallest party that requested to abort in π. Then:
   (a) If abort because $P_{i'}$ sent an inconsistent message, P_i broadcasts $(\mathsf{corrupt}, i')$. All parties output $(\mathsf{corrupt}, i, i')$.
   (b) If abort because P_i detected inconsistency in an inner protocol session j:
      i. P_i broadcasts $(\mathsf{corruptsession}, j)$.
      ii. All parties broadcast their views in session j.
      iii. If there exists inconsistency in the views, everyone identifies the lexicographically smallest pair $\{P_{i_1}, P_{i_2}\}$ with inconsistency and outputs $(\mathsf{corrupt}, i_1, i_2)$
      iv. Else, P_i detects a party $P_{i'}$ whose view as revealed now is different from its view as revealed over the watch-list channel and broadcasts $(\mathsf{corrupt}, i')$; everyone outputs $(\mathsf{corrupt}, i, i')$.

(c) If P_i does not carry out either of the above steps, everyone outputs $(\mathsf{corrupt}, i, i')$, where i' is the lexicographically smallest active party other than i.

F.4 Communication Cost Analysis

Here we analyze the cost of the protocol obtained from the simpler BBT used to prove Theorem 5. For this, we consider the communication and randomness cost of all the sessions of the inner protocol (i.e., the cost in the protocol generated by $\mathrm{IPS}_{\mathrm{core}}$) and the communication cost of the watchlist setup separately.

The communication and randomness costs in all the inner protocol sessions together is similar to that in the IPS transformation. If the outer protocol scheme is a $(p_{\mathrm{out}}, q_{\mathrm{out}})$-$\Lambda_{\beta\text{-}\mathsf{full}}$ scheme then, referring to the notation in Table 3, the communication cost of the outer protocol would be $O(p_{\mathrm{out}}(n,k) \cdot \mathsf{size}(f) + \mathrm{poly}(n,k))$. the computation cost would be $O(q_{\mathrm{out}}(n,k) \cdot \mathsf{size}(f) + \mathrm{poly}(n,k))$. Since the IPS compiler uses an additive secret-sharing to encode the communication between the servers and share it among the n clients, the total computation that is implemented by the inner protocol sessions is $O(q'(n,k) \cdot \mathsf{size}(f) + \mathrm{poly}(n,k))$, where $q' = q_{\mathrm{out}} + n \cdot p_{\mathrm{out}}$. If the inner protocol scheme is a $(p_{\mathrm{in}}, q_{\mathrm{in}}, r_{\mathrm{in}})$-$\Lambda_{\alpha\text{-}\mathsf{sh}}$ scheme, then this translates to a total communication plus randomness cost in inner protocol sessions of $O((p_{\mathrm{in}}(n,k) + r_{\mathrm{in}}(n,k)) \cdot q'(n,k) \cdot \mathsf{size}(f) + \mathrm{poly}(n,k))$.

On top of this, the watchlist mechanism using w^* imposes a $O(n^2)$ factor communication cost. This is because, each bit of randomness and communication in the inner protocol sessions need to be transmitted to all n parties over the watchlist channel,[16] and each bit communicated between a pair of parties over the watchlist channel corresponds to $O(n)$ bits of communication in the protocol $\pi_{\widetilde{\mathsf{OT}}}$. Thus, the overall communication cost is $O(p(n,k) \cdot \mathsf{size}(f) + \mathrm{poly}(n,k))$, where $p = (p_{\mathrm{in}} + r_{\mathrm{in}})(q_{\mathrm{out}} + n \cdot p_{\mathrm{out}})$.

There is an additional communication cost introduced by T_{id}, due to all parties broadcasting their view of one inner protocol session during the identification procedure. But this added cost cannot be larger than the communication and randomness cost of a single session of the inner protocol. Hence this transformation does not alter the asymptotic cost.

G Analysis for Theorem 6

Recall that in the BBT from $\{\Lambda_{\beta\text{-}\mathsf{full}}, \Lambda_{\alpha\text{-}\mathsf{sh}}\}$ to $\Lambda_{\alpha\text{-}\mathsf{id/BC}}$, we use the same watchlist setup functionality $\mathcal{W}$ as in the IPS transformation (but further modified to $\mathcal{W}^*$ to enforce partially-identifiable abort). In $\mathcal{W}$ (and hence in $\mathcal{W}^*$), each server's execution (carried out by a session of the inner protocol) can be "potentially watched" by every party P_i. Thus, each party taking part in an inner protocol session will have to send its view to all the other parties over the watchlist channels, leading to a multiplicative overhead of n. However, we note that it is sufficient to have each server potentially watched by at least one honest party. (Indeed, in the analysis in [23], the number of parties could just be 2, and if one of them is corrupt, then each server is watched by only one honest party.) Given that we have a guarantee that at most $\alpha n \leq n/2$ parties are corrupted, we can use a much sparser graph (rather than the complete bipartite graph) to define which parties watch can potentially watch which servers' executions.

Towards this, we use an expander graph between the set of parties and the set of servers in the outer protocol, in which the degree of each server is a constant, but any subset of $n/2$ parties will potentially watch almost all of the servers. For our purposes, we define an expander graph as follows.

[16] The motivation for the streamlined protocol in Theorem 6 is to save this factor of n in setting up the watchlists, by exploiting an honest-majority.

Definition G.1 ($(n, n'; N, N'; \bar{D})$-Expander)**.** *Let $G \subseteq [n] \times [N]$ and for every $j \in [n]$, let $\Gamma(j) = \{\ell \in [N] | (j, \ell) \in G\}$. We say that G is an $(n, n'; N, N'; \bar{D})$-Expander if for every $j \in [n]$, we have $|\Gamma(j)| \leq \bar{D}$, and for every $S \subseteq [n]$ with $|S| = n'$, we have $|\bigcup_{j \in S} \Gamma(j)| \geq N'$.*

It follows from standard results (see, e.g., [32]), that there are explicit constructions with the following parameters.

Lemma 1. *For all $0 < \delta < 1$, there is a constant c_δ such that for all but finitely integers $n > 0$, $N \geq n$, there is an (explicit) $(n, n/2; N, (1 - \delta/2)N; \bar{D})$-expander, where $\bar{D} = c_\delta N/n$.*

From the above lemma we can obtain an $(n, n/2; N, (1 - \beta/2)N; \bar{D})$-expander $G_{n,k,\beta}$ where $N = kn^2$ and $\bar{D} = cN/n$ (for some constant c and sufficiently large n).

To use the sparse watchlists in our transformation modularly, we modify the functionality $\mathcal{W}$ to be parametrized by a graph G which it uses to define the set of one-time pads needed. For the sake of completeness, this modified functionality, $\mathcal{W}^+$ is shown in Appendix G.1.

Now we analyze the communication complexity of the protocol scheme obtained using this modified transformation. The dominant cost is that of the watchlist setup protocol. Note that the communication plus randomness complexity of all the inner protocol sessions together, $K^* = \tilde{O}((p_{\text{in}} + r_{\text{in}})(q_{\text{out}} + n \cdot p_{\text{out}}))$ (ignoring lower-order terms). The total size of the one-time pads generated by $\mathcal{W}^+$ is $\sum_{\ell=1}^{N} K d_\ell$, where K is the communication plus randomness cost for one inner-protocol session and d_ℓ is the degree of the ℓ^{th} server's vertex in the expander graph G.[17] But $\sum_\ell d_\ell = cN$, and since $K^* = NK$, we have that the total size of the one-time pads is $O(K^*) = \tilde{O}((p_{\text{in}} + r_{\text{in}})(q_{\text{out}} + n \cdot p_{\text{out}}))$. Hence the dominating communication cost in the protocol, which results from the invocations of $\pi_{\widetilde{\text{OT}}}$ is $\tilde{O}(n \cdot (p_{\text{in}} + r_{\text{in}})(q_{\text{out}} + n \cdot p_{\text{out}}))$.

We point out that the choice $N = k^2 n$ and the transmission probablity $\gamma = \beta/(cn)$ can meet the security except an exponentially low probability. This follows by relating to the analysis in [23] as follows. If the adversary who corrupts $n/2 - 1$ parties corrupts (deviate in or attempt to "watch" the inner-protocol execution of) up to $(n/2)^2 \gamma \bar{D} < \beta N$ servers so the information is protected by the outer protocol. If she actively corrupts more than β fraction of the servers, then by the expansion property of the graph G, at least $(\beta/2)kn^2$ of them are potentially watched by at least one honest client. Hence the corruption will be identified by at least one honest client, with probability at least $1 - \exp(-\Omega(kn))$.

G.1 Formal Description of T'_3 and $\mathcal{W}^+$

Transformation T'_3: Given a set of protocols $\pi^t_{\alpha\text{-id}} t \in [n]$ (each of which parametrized by number of parties) and an ECSS scheme $(\mathsf{share}, \mathsf{reconstruct})$:

1. Set the set of active parties $T^* = [n]$.
   (a) Set `done = false`.
   (b) While `done` is `false`
      i. $\forall i \in [n]$, $\mathcal{P}_i$ shares its input x_i as $(\sigma^i_{j_1}, \cdots, \sigma^i_{j_{n'}}) \leftarrow \mathsf{share}(x_i, n')$ where $T^* = \{j_1, \cdots, j_{n'}\}$; it sends σ^i_j to $\mathcal{P}_j$, for all $j \in T^*$.
      ii. Parties in T^* run the protocol $\pi_{\alpha\text{-id}}[n']$, with $\mathcal{P}_j$'s input being $(\sigma^1_j, ..., \sigma^n_j)$ (using $\perp$ for any σ^i_j that was not received).

[17] For simplicity, here, as in [23], we consider all server executions to have the same complexity. But even if there is no load-balancing, we can obtain an efficient construction by assigning lower degree vertices in the expander to servers with higher complexity. Indeed, even if the complexity changes dynamically, one can adapt the watchlist setup protocol to adaptively extend the one-time pads. We describe these generalizations in the full version.

iii. For each $j \in T^*$, if $\mathcal{P}_j$ receives the output $(\mathsf{corrupt}, \ell_1, \ell_2)$ from the above protocol, then it lets $T^* = T^* \setminus \{\ell_1, \ell_2\}$ and broadcasts T^*; else $\mathcal{P}_j$ gets output $(\delta_j^1, ..., \delta_j^n)$ and broadcasts "done."

iv. For each $i \in [n]$, on receiving "done" from $n'/2$ or more parties in T^*, $\mathcal{P}_i$ sets `done = true`; else it sets T^* to be what was broadcast by a majority of parties in T^*.

(c) For each $i \in [n]$, $j \in T^*$, $\mathcal{P}_j$ sends δ_j^i to $\mathcal{P}_i$.

(d) For each $i \in [n]$, $\mathcal{P}_i$ receives $(\delta_1^i, ..., \delta_{n'}^j)$ and sets $x_i = \mathsf{reconstruct}(\delta_1^i, ..., \delta_{n'}^i)$.

2. For each $i \in [n]$, $\mathcal{P}_i$ outputs x_i.

Sparse Watchlist Setup Functionality $\mathcal{W}^+$. An n-party functionality $\mathcal{W}^+[K, d, \delta, N, G]$ parametrized by the length of pads (communication complexity of the watched protocol) K, $d \in [n]$ such that $\gamma = \frac{1}{d}$ is the expected fraction of pads to be delivered (i.e., fraction of servers to be watched by an honest party), $0 \leq \delta < 1$ being the fraction of servers for which the adversary can learn if a pad corresponding to it was delivered to any honest party or not, the number of servers N, and $G \subseteq [n] \times [N]$ (denoting the edges of a bipartite graph with partite sets $L = [n]$ and $R = [N]$). The trusted party interacts with the honest parties, the corrupted parties, and the adversary Adv as follows. Below, for $j \in [n]$, let $\Gamma(j) = \{\ell \in [N] : (j, \ell) \in G\}$.

1. For each honest party P_i, for each $\boxed{(j, \ell) \in G}$, pick a random K-bit string $\mathsf{Pad}_\ell^{i \to j}$ and give it to P_i.

2. For each honest party P_j, pick random $\boxed{L^j \subseteq \Gamma(j)}$, $|L^j| = \gamma N$, and send $\{\mathsf{Pad}_\ell^{i \to j} | \ell \in L^j, i \in [n]\{j\}$ to P_j.

3. From each corrupted party P_i, for each $\boxed{\ell \in \Gamma(i)}$ and $j \in [n]$, accept a K-bit string $\mathsf{Pad}_\ell^{i \to j}$.

4. From each corrupted party P_j, accept a set of pairs $S^j \subseteq \{(i, \ell) | i \in [n], \ell \in [N]\}$ such that $\forall i \in [n], S_i^j = \{\ell | (i, \ell) \in S^j\}$ is of size $|S_i^j| \leq 2\gamma N$. Send $\{\mathsf{Pad}_i^{i \to j} | (i, \ell) \in S^j\}$ to P_j for each corrupted P_j.

   $\star$ Also, accept a set $S \subseteq [n]$, $|S| \leq \delta N$ from the adversary. For each honest party P_i, reveal $S \cap L^i$ to the adversary.

5. At any point Adv can ask the trusted party to send abort to any (honest) party.
