SECURE TRANSFORMATION OF CRYPTOGRAPHIC PROTOCOLS

BY

CHING-HUA YU

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2020

Urbana, Illinois

Doctoral Committee:

Adjunct Professor Manoj Prabhakaran, Director of Research
Professor Jeff Erickson, Chair
Assistant Professor Andrew Miller
Professor Yuval Ishai, Technion IIT

ABSTRACT

Over decades of active research, MPC has grown into a rich and complex topic, with many incomparable flavors and numerous protocols and techniques. The 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. Meanwhile, several important results actually rely on "protocol transformations," whereby protocols from one model of MPC are transformed to protocols from another model.

Motivated by simplifying and unifying results in the area of MPC, our first goal is to formalize a general notion of black-box protocol transformations that captures previous transformations from the literature as special cases, and present several new transformations. In addition to the simplification of known feasibility results, we then push our study of protocol transformations by presenting several results regarding security augmentation and efficiency leveraging. On the other hand, we prove the *impossibility* of two simple types of black-box protocol transformations.

Next, we initiate the study of *bottleneck* complexity as a new communication efficiency measure for secure multiparty computation (MPC). Roughly, the bottleneck complexity of an MPC protocol is defined as the maximum communication complexity required by any party within the protocol execution. While achieving $O(n)$ bottleneck complexity (where n is the number of parties) is straightforward, we show that: (1) achieving *sublinear* bottleneck complexity is *not* always possible, even when no security is required. (2) On the other hand, several useful classes of functions do have $o(n)$ bottleneck complexity, when no security is required.

Then our main positive result regarding *bottleneck* complexity is a compiler that transforms any (possibly insecure) efficient protocol with fixed communication-pattern for computing any functionality into a secure MPC protocol while preserving the bottleneck complexity of the underlying protocol (up to security parameter overhead). Given our compiler, an efficient protocol for any function f with sublinear bottleneck complexity can be transformed into an MPC protocol for f with the same bottleneck complexity.

Along the way, we build cryptographic primitives – incremental fully-homomorphic encryption, succinct non-interactive arguments of knowledge with ID-based simulation-extractability property and verifiable protocol execution – that may be of independent interest.

To my parents, for their love and support.

ACKNOWLEDGMENTS

I am so fortunate to be advised by Professor (Prof.) Manoj Prabhakaran. He a very intelligent and hard-working researcher and in the meantime a really patient and wise mentor. I had started the research in cryptography before entering University of Illinois at Urbana-Champaign (UIUC). The learning track was like collecting unintegrated fragments from literature and gathering sporadic scatters from people in the fields. It was not until the projects with Manoj that I had a chance to see a more complete picture and learn how people doing in the field, especially when he walked through each research stage with me. The skills and the philosophy I learned from him are useful even beyond the cryptography study.

I appreciate each cryptography project that I have participated in and the people that I have worked with. In alphabetical order, they are Shashank Agrawal, Elette Boyle, Kai-Min Chung, Yuval Ishai, Abhishek Jain, Eyal Kushilevitz, Feng-Hao Liu, and Amit Sahai. The collaboration with these amazing people is the best memory in my Ph.D. program.

I want to give a thank to Prof. Abhishek Jain for hosting an academic visit for me in Johns Hopkins University. In our project with Abhishek and Prof. Elette Boyle, I started on some cryptographic topics fresh to me and enjoyed the creative constructions together. From their generous sharing, I learned not only skills and knowledge but also enthusiasm and optimism in doing research. I appreciate Prof. Kai-Min Chung and Prof. Fen-Hou Liu for the past projects and the sharing of their experience in research as well as in life. Together with Prof. Yu-Chi Chen, Wei-Kai Lin and many others, the months we spent to discuss cryptographic topics in coffee shops and bars in Berkeley and in New York City were memorable. I want to give a thank to Prof. Yuval Ishai also. I was impressed every time by his endless erudition and fast responses in our discussion. The works with them also demonstrate to me the great part of the academic interactions between scholars in the field.

In addition, I want to thank Professors Jeff Erickson, Yuval Ishai, and Andrew Miller for serving on my Ph.D. committee and for their precious feedback and discussion.

The research atmosphere of our theory group in UIUC has been very promoted, and the interactions between the members have been a pleasure. I would like to thank Prof. Chandra Chekuri and the other faculty members in our group, for their effort in holding events, arranging and encouraging students to attend local as well as global academic activities and their cares for the students. In addition, our group members, Calvin Beideman, Hsien-Chih Chang, Ziwei Ji, Pooja Kulkarni, Rucha Kulkarni, Vasilis Livanos, Ian Ludden, Kent

Quanrud, Yipu Wang, Chao Xu, and others have all been very nice and helpful.

I sincerely appreciate the great experiences of teaching assistantships in UIUC. I have a wonderful TA'ing time in CS374 (Introduction to Algorithms & Models of Computation) taught by Prof. Jeff Erickson, another CS374 taught by Prof. Timothy Moon-Yew Chan and Prof. Ruta Mehta, CS498AML (Applied Machine Learning) taught by Prof. Trevor Walker, CS498DV (Data Visualization) taught by Prof. John Hart, and CS447 (Natual Language Processing) taught by Prof. Julia Hockenmaier. Especially they gave me a chance not only in practicing teaching but also in observing the course design ideas and structures in the department as well as the guideline and expectation they have for the students. Their teaching ideal and enthusiasm have affected me deeply.

Besides, it is my honor to work with other TAs, Calvin Beideman, Pooja Kulkarni, Rucha Kulkarni, Vasilis Livanos, Ian Ludden, Keval Morabia, Wendi Shi, Tana Wattanawaroon, Peiye Zhuang, Shiliang Zuo, and many others. They have been very friendly and active when we were designing questions, preparing labs, and resolving course issues together.

Last but not least I want to thank my family. They have been very supportive in the back this long time and always encourage me when I am pursuing my goals. I wouldn't go this far without them, and I dedicate this milestone to them.

TABLE OF CONTENTS

CHAPTER 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 approaches (e.g., simulation-based security [1, 2]), powerful and vastly applicable algorithmic techniques (starting with secret-sharing [3] and garbling schemes [4]), sharp impossibility results (e.g., [5]) and even several cryptographic concepts ahead of their time (like fully-homomorphic encryption [6]). 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. [7], which proposes classifying the existing protocols using 22 dimensions.

1.1 TRANSFORMATION OF MPC MODELS

The diversity of MPC 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 [7] 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 [2, 8, 9, 10, 11, 12], 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 [11, 13] are asymptotically less efficient compared to similar protocols with security against $t < n/3$ parties [9], or even to protocols that offer "security with abort" against $t < n$ malicious parties [14].

A classical example of a general protocol transformation is the well known "GMW compiler," [2], 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" [15] and the IPS transformation that combines player-virtualization with "watchlists" [14]. 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 [11] regarding security with guaranteed output delivery with an honest majority, from two simpler feasibility results (both of which were solved in [9, 10]): (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 [17]. In Section 3.3.1, 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 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

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

of efficiency leveraging is given by Bracha's transformation [18], 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.

1.2 A NEW MEASURE OF COMMUNICATION EFFICIENCY

Next, we initiate the study of *bottleneck* complexity as a new communication efficiency measure for secure multiparty computation (MPC). Roughly, the bottleneck complexity of an MPC protocol is defined as the maximum communication complexity required by any party within the protocol execution.

We observe that even without security, bottleneck communication complexity is an interesting measure of communication complexity for (distributed) functions and propose it as a fundamental area to explore. While achieving $O(n)$ bottleneck complexity (where n is the number of parties) is straightforward, we show that: (1) achieving *sublinear* bottleneck complexity is *not* always possible, even when no security is required. (2) On the other hand, several useful classes of functions do have $o(n)$ bottleneck complexity, when no security is required.

Our main positive result is a compiler that transforms any (possibly insecure) efficient protocol with fixed communication-pattern for computing any functionality into a secure MPC protocol while preserving the bottleneck complexity of the underlying protocol (up to security parameter overhead). Given our compiler, an efficient protocol for any function f with sublinear bottleneck complexity can be transformed into an MPC protocol for f with the same bottleneck complexity.

Along the way, we build cryptographic primitives – incremental fully-homomorphic encryption, succinct non-interactive arguments of knowledge with ID-based simulation-extractability property and verifiable protocol execution – that may be of independent interest.

Following these lines, the content of this thesis divide into the following parts. In Chapter 2, we review the background of MPC involving some existing techniques that will be used in the remaining chapters. In Chpater 3, we propose a black-box transformation (BBT) framework of MPC protocols. In Chpater 4, we provide new BBT results involving security augmentation and efficiency leveraging. In Chapter 5, we model bottleneck communication complexity as a new efficiency measurement. In Chpater 6, we develop secure transformation that preserves bottleneck communication complexity. Finally in Chapter 7, we summarize the results and discuss future works.

CHAPTER 2: MPC BACKGROUND

We review the background of secure multiparty computation as well as some cryptographic techniques that will be used later in the remaining chapters. Particularly, the definition and background described in Section 2.1, 2.2, 2.3, 2.4 and 2.5 will be used in Chapter 3 and 4, and those in Section 2.6 will be used in Chapter 5 and 6.

2.1 PROTOCOL AND FUNCTIONALITY

Communication Model. We consider a synchronous network among n parties, $P_1, \cdots, P_n$, that allows secure communication between (some) pairs of parties; the channels are authenticated and leak nothing except the number of bits in each message.

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 2.1 (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,[1] 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.

[1] 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.

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.

2.2 SECURITY

$\Lambda^{\mathcal{F}}_{\mathsf{secure}}$	(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 2.1: Terminology used for guarantees from protocols.

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

$$\Lambda^{\mathcal{F}}_{\mathsf{secure}} = \{(f, \pi) \mid (f^{\langle r \rangle}, \pi) \in \widehat{\Lambda}^{\mathcal{F}^+}_{\mathsf{secure}} \text{ for some polynomial } r\}. \tag{2.1}$$

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

In Table 2.1 we name some of the main security definitions considered in our results. For instance, $\Lambda^{\mathcal{F}}_{\alpha\text{-}\mathsf{full}}$ 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 $\Lambda_{\alpha\text{-}\mathsf{abort}}$ as

$$\Lambda_{\alpha\text{-}\mathsf{abort}} = \{(f, \pi) \mid (f^{\langle \mathsf{abort} \rangle}, \pi) \in \Lambda_{\alpha\text{-}\mathsf{full}}\}. \tag{2.2}$$

Stand-Alone Security. This is essentially the model of [19], but we present it as a restriction to the UC security definition. Specifically, we define a *standalone environment* to be one which initiates a single session (of Π of $\mathcal{F}$), and does not communicate with the adversary until all the honest parties terminate. That is, a standalone environment can only interact with the adversary prior to the start of the protocol, and after it terminates.

We say that a protocol Π is a *standalone-secure* protocol for a functionality $\mathcal{F}$ if it UC-securely realizes $\mathcal{F}$ (with selective abort) when restricted to standalone environments. Here, in the ideal model, the adversary is allowed to cause individual honest parties to abort, after obtaining the outputs for the corrupt parties. We point out that the definition of UC-security allows a *non-black-box* simulator that depends on the adversary. (Unlike in UC-security, existence of a simulator in the standalone setting does not imply the existence of a black-box simulator. In UC-security, one may replace the adversary with a dummy adversary which interacts with the actual adversary which is kept inside the environment; in the standalone setting the dummy adversary cannot interact with the environment during

the protocol execution.)

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). This notion of security generalizing the notion of security with identifiable abort and will be defined in detail later.

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). 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.

2.3 PROTOCOL SCHEME AND COMPLEXITY

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

Definition 2.2 (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 $\mathcal{P}(f)$.

2.3.1 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 $p\text{-}\Lambda^{\mathcal{F}}_{\mathsf{secure}}$ 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

$$O(p(n,k) \cdot \mathsf{size}(f) + \mathrm{poly}(n,k)). \tag{2.3}$$

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: $(p,q,r;\mathsf{D})\text{-}\Lambda^{\mathcal{F}}_{\mathsf{secure}}$ 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* [20], 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 $\langle\gamma,\delta\rangle\text{-}\Lambda_{\alpha\text{-}\mathsf{id}}$ 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 $(p,q,r;\mathsf{D})\text{~}\Lambda^{\mathcal{F}}_{\mathsf{secure}}$ instead of $(p,q,r;\mathsf{D})\text{-}\Lambda^{\mathcal{F}}_{\mathsf{secure}}$ and so on, if we intend to use $\widetilde{O}(\cdot)$ instead of $O(\cdot)$ in the above costs.[2] The notation is summarized in Table 2.2.

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

$(p, q, r; \mathsf{D})$ - $\Lambda_{\mathsf{secure}}$	$\Lambda_{\mathsf{secure}}$ 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 2.2: Additional notation for protocol schemes (for n parties, and security parameter k).

2.4 SECRET SHARING

2.4.1 Additive Secret Sharing

By $x \in_R \mathbb{Z}_q^n$ we denote that x is uniformly sampled from $\mathbb{Z}_q^n$, and by $x \leftarrow D$ we denote that x is sampled from a distribution D. By $\overset{c}{\approx}$ we denote computational indistinguishability. We denote an N-party additive secret sharing of $x \in \mathbb{Z}_q$ by $[x]_q^N$. That is, each P_i owns $x_i \in \mathbb{Z}_q$ such that $x = \sum_{i \in [N]} x_i$. When it is clear, we write $[x]$ for brevity.

2.4.2 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 [11, 21, 22]. 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 below.

Definition 2.3 (Error-Correcting Secret Sharing)**.** A pair of algorithms (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 $\mathsf{reconstruct}(\sigma'_1, ..., \sigma'_n) = s$.

2.5 A SIMPLE ERROR CORRECTING SECRET-SHARING SCHEME

We defined ECSS in Section 2.4.2. 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 [22] improving on previous schemes [11, 21].

ECSS $-$ 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 $\tau_{i,j} = \text{MAC}_{K_{i,j}}(\rho_j) \forall i, j \in [n], i \neq j$
4. For all i, set $\sigma_i = (\rho_i, \{K_{i,j} | j \neq i\}, \{\tau_{j,i} | j \neq i\})$
5. Output $(\sigma_1, ..., \sigma_n)$

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

1. Define $\mathsf{consistent}(\sigma_i, \sigma_j) =$ True iff $\tau_{i,j} = \text{MAC}K_{i,j}(\rho_j)$ and $\tau_{j,i} = \text{MAC}_{K_{j,i}(\rho_i)}$.
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.

2.6 CRYPTOGRAPHIC PRIMITIVES

2.6.1 Multisignatures

In a multisignature scheme, a single short object—the *multisignature*—can take the place of n signatures by n signers, all on the same message.[3] The first formal treatment of multisignatures was given by Micali, Ohta, and Reyzin [24]. We consider a variant of the Micali-Ohta-Reyzin model due to Boldyreva [25], as presented in [26]. In this model, the adversary is given a single challenge verification key VK, and a signing oracle for that key.

[3]Note that multisignatures are a special case of *aggregate* signatures [23], which in contrast allow combining signatures from n different parties on n *different* messages.

His goal is to output a forged multisignature σ^* on a message m^* under keys $\text{VK}_1, ..., \text{VK}_\ell$, where at least one of these keys is a challenge verification key (wlog, VK_1). For the forgery to be nontrivial, the adversary must not have queried the signing oracle at m^*.

For simplicity, we present a slightly weaker version of the security definition achieved by [26], which suffices for our application.[4]

Definition 2.4. A *multisignature* scheme is a tuple of algorithms

$\mathsf{KeyGen}(1^k)$**:** Key generation algorithm. Outputs a secret signing key SK together with corresponding public verification key VK.

$\mathsf{Sign}(\text{SK}, m)$**:** Standard signing algorithm, with respect to message m and single signing key SK.

$\mathsf{Combine}(\{\text{VK}_i, \sigma_i\}_{i=1}^{\ell}, m)$**:** Takes as input a collection of signatures (or multisignatures) and outputs a combined multisignature, with respect to the union of verification keys.

$\mathsf{MultiVer}(\{\text{VK}_i\}_{i=1}^{\ell}, m, \sigma)$**:** Verifies multisignature σ with respect to the collection of verification keys $\{\text{VK}_i\}_{i=1}^{\ell}$. Outputs 0 or 1.

that satisfies the following properties:

Correctness: For any message m, any collection of honestly generated signatures $\{\sigma_i \leftarrow \mathsf{Sign}_{\text{SK}_i}(m)\}_{i\in I}$ on m (for $I \subset [n]$), the combined multisignature formed by $\bar{\sigma} \leftarrow \mathsf{Combine}(\{\text{VK}_i, \sigma_i\}_{i\in I}, m)$ will properly verify with overwhelming probability: $\Pr[1 \leftarrow \mathsf{MultiVer}(\{\text{VK}_i\}_{i\in I}, m, \bar{\sigma})] \geq 1 - \text{negl}(k)$.

Unforgeability: For any PPT adversary $\mathcal{A}$, the probability that the challenger outputs 1 when interacting with $\mathcal{A}$ in the following game is negligible in the security parameter k:

Setup. The challenger samples n pulic key-secret key pairs, $(\text{VK}_i, \text{SK}_i) \leftarrow \mathsf{KeyGen}(1^k)$ for each $i \in [n]$, and gives $\mathcal{A}$ all verification keys $\{\text{VK}_i\}_{i\in[n]}$. $\mathcal{A}$ selects a proper subset $M \subset [n]$ (corresponding to parties to corrupt) and receives the corresponding set of secret signing keys $\{\text{SK}_i\}_{i\in M}$.

Signing queries. $\mathcal{A}$ may make polynomially many adaptive signature queries, of the form (m, VK_i). For each such query, the challenger responds with a signature $\sigma \leftarrow \mathsf{Sign}_{\text{SK}_i}(m)$ on message m with respect to the corresponding signing key SK_i.

[4]The security game in [26] also allows the adversary the power to choose verification keys on behalf of corrupted parties, as long as he also provides certification that the keys were properly generated.

Output. A outputs a triple $(\bar{\sigma}^*, m^*, \{\mathrm{VK}_i\}_{i \in S})$, where $\bar{\sigma}^*$ is an alleged forgery multisignature on message m^* with respect to a subset of verification keys $S \subset [n]$. The challenger outputs 1 if at least one of the provided verification keys VK_i corresponds to a challenge (honest party) key, the message m^* was not queried to the signature oracle with this verification key VK_i, and the provided forgery σ^* is a valid multisignature: i.e., $1 \leftarrow \mathsf{MultiVer}(\{\mathrm{VK}_i\}_{i \in S}, m^*, \sigma^*)$.

The following theorem follows from a combination of the (standard) signature scheme of Waters [27] together with a transformation from this scheme to a multisignature scheme due to Lu et. al. [26].

Theorem 2.1 ([26])**.** There exists a secure multisignature scheme with signature size $\mathrm{poly}(k)$ (independent of message length and number of potential signers), based on the Bilinear Computational Diffie-Hellman assumption.

For convenience of notation, we shall use a multisignature scheme also as a normal signature scheme. In that case, we shall write $\mathsf{MS.Verify}(\mathrm{VK}_i, m, \sigma)$ instead of $\mathsf{MS.MultiVer}(\{\mathrm{VK}_i\}, m, \sigma)$ to indicate that the set of keys involved is singleton.

Remark 2.1. We note that in our constructions, we can instantiate a multisignature scheme with a simulation-extractable zero-knowledge SNARK with additive overhead (defined below) and standard signatures.

2.6.2 Succinct Non-Interactive Arguments of Knowledge

We consider succinct non-interactive arguments *of knowledge* (SNARKs) with adaptive soundness. Our treatment follows that of Bitansky *et al.* [28]. We focus attention to *publicly verifiable* succinct arguments. Due to recent results demonstrating implausibility of SNARKs with respect to arbitrary worst-case auxiliary input (e.g., [29, 30]), we consider a definition parameterized with respect to a particular auxiliary input distribution $\mathcal{Z}$.

Definition 2.5 ($\mathcal{Z}$-auxiliary input SNARK)**.** A triple of algorithms $(\mathsf{crsGen}, \mathsf{Prove}, \mathsf{Verify})$ is a *publicly verifiable, adaptively sound succinct non-interactive argument of knowledge (SNARK)* for the relation $\mathcal{R}$ with respect to auxiliary input distribution $\mathcal{Z}$ if the following conditions are satisfied for security parameter λ:

- **Completeness:** For any $(x, w) \in \mathcal{R}$,

$$\Pr[\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\lambda); \pi \leftarrow \mathsf{Prove}(x, w, \mathsf{crs}) : \mathsf{Verify}(x, \pi, \mathsf{crs}) = 1] = 1. \tag{2.4}$$

In addition, $\mathsf{Prove}(x, w, \mathsf{crs})$ runs in time $\mathrm{poly}(\lambda, |x|, t)$.

- **Succinctness:** The length of the proof π output by $\mathsf{Prove}(x, w, \mathsf{crs})$, as well as the running time of $\mathsf{Verify}(x, \pi, \mathsf{crs})$, is bounded by $p(\lambda, |X|)$, where p is a universal polynomial that does not depend on $\mathcal{R}$. In addition, $\mathsf{crsGen}(1^\lambda)$ runs in time $\mathrm{poly}(\lambda)$: in particular, crs is of length $\mathrm{poly}(\lambda)$.

- **Adaptive Argument of Knowledge:** For any non-uniform polynomial-size prover P^* there exists a polynomial-size extractor $\mathcal{E}_{P^*}$, such that for all sufficiently large $\lambda \in \mathbb{N}$ and auxiliary input $z \leftarrow \mathcal{Z}$, it holds that

$$\Pr[z \leftarrow \mathcal{Z};\ \mathsf{crs} \leftarrow \mathsf{crsGen}(1^\lambda);\ (x, \pi) \leftarrow P^*(z, \mathsf{crs});\ (x, \pi, w) \leftarrow \mathcal{E}_{P^*}(z, \mathsf{crs}) : \mathsf{Verify}(x, \pi, \mathsf{crs}) = 1 \wedge w \notin R(x)] \leq \mathrm{negl}(\lambda). \quad (2.5)$$

Extraction with Additive Overhead. We also consider SNARKs where the extractor incurs only an additive overhead in the running time of the adversarial prover. A $\mathcal{Z}$-auxiliary-input SNARK is said to satisfy the *additive overhead extraction* property if there exists a polynomial p such that for all polynomial time P^*, there exists an $\mathcal{E}_{P^*}$ as in Definition 2.5, such that for all z in the support of $\mathcal{Z}$ and all crs in the support of crsGen,

$$\mathrm{RT}(\mathcal{E}_{P^*}(z, \mathsf{crs})) \leq p(\lambda) + \mathrm{RT}(P^*(z, \mathsf{crs})), \quad (2.6)$$

where $\mathrm{RT}(A)$ denotes the running time of an algorithm A.

2.6.3 GSW FHE Scheme

We follow the notation of [31] throughout this section. We start by recalling some preliminary definitions and then present the FHE scheme of Gentry, Sahai and Waters [32]. We use their FHE scheme as a key building block in our IFHE scheme.

LWE assumption. We first recall the learning with errors assumption [33].

Definition 2.6 (LWE Hardness Assumption)**.** Let λ be a security parameter, $\chi = \chi(\lambda)$ be a distribution of small values over $\mathbb{Z}$, $n = n(\lambda)$ and $q = q(\lambda)$ be polynomials of λ, and $m = O(n \log q)$. Let $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^{n-1}$, $\mathbf{B} \xleftarrow{\$} \mathbb{Z}_q^{(n-1)\times m}$, $\mathbf{e} \leftarrow \chi$, and $\mathbf{b} = \mathbf{sB} + \mathbf{e}$. Then $(\mathbf{B}, \mathbf{b}) \overset{c}{\approx} (\mathbf{B}', \mathbf{b}')$, where $(\mathbf{B}', \mathbf{b}') \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$.

Public Short Preimage Matrix. We state a useful fact from [34] that is used in the GSW FHE scheme.

Lemma 2.1 ([34]). For any $m \geq n(\lfloor \log q \rfloor + 1)$, there is a gadget matrix $\mathcal{G} \in \mathbb{Z}_q^{n \times m}$ and an efficient deterministic function $\mathcal{G}^{-1}(\cdot)$ such that for any m', any $\mathbf{M} \in \mathbb{Z}_q^{n \times m'}$, $\mathcal{G}^{-1}(\mathbf{M}) \in \{0,1\}^{m \times m'}$, and $\mathcal{G}\mathcal{G}^{-1}(\mathbf{M}) = \mathbf{M}$.

In the GSW scheme, the function $\mathcal{G}^{-1}(\cdot)$ is called $\mathsf{BitDecomp}$ and multiplication by $\mathcal{G}$ is the $\mathsf{BitDecomp}^{-1}$ operation. For our purposes, we do not need their implementation details.

GSW Construction. We now proceed to describe the GSW FHE scheme.

- **Setup:** $(\mathsf{params}) \leftarrow \mathsf{GSW.Setup}(1^\lambda, 1^d)$
  Choose a lattice with dimension parameters $n = n(\lambda, d)$, B_χ-bounded error distribution $\chi = \chi(\lambda, d)$ and a modulus q such that $\mathsf{LWE}_{n-1,q,\chi,B_\chi}$ holds. Choose $m = O(n \log q)$. Finally, choose a random matrix $\mathbf{B} \in \mathbb{Z}_q^{n-1 \times m}$. Output $\mathsf{params} = (q, n, m, \chi, B_\chi, \mathbf{B})$.

- **Key Generation:** $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{GSW.Keygen}(\mathsf{params})$
  We separately describe two sub-algorithms to compute secret-key and public-key respectively:

  - $\mathsf{GSW.SKGen}(\mathsf{params})$: Sample $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^{n-1}$. Set $\mathbf{t} = (-\mathbf{s}, 1) \in \mathbb{Z}_q^n$ and output $\mathrm{SK} = \mathbf{t}$.
  - $\mathsf{GSW.PKGen}(\mathsf{params}, \mathrm{SK})$: Parse $\mathrm{SK} = (-\mathbf{s}, 1) \in \mathbb{Z}_q^n$. Sample $\mathbf{e} \leftarrow \chi^m$. Set $\mathbf{b} = \mathbf{s}\mathbf{B} + \mathbf{e} \in \mathbb{Z}_q^m$ and $A = \begin{bmatrix} \mathbf{B} \\ \mathbf{b} \end{bmatrix} \in \mathbb{Z}_q^{n \times m}$. Output $\mathrm{PK} = \mathbf{A}$.

- **Encryption:** $C \leftarrow \mathsf{GSW.Encrypt}(\mathrm{PK}, x)$
  On input a message $x \in \{0,1\}$, choose a short random matrix $\mathbf{R} \xleftarrow{\$} \{0,1\}^{m \times m}$ and compute $\mathbf{C} = \mathbf{A}\mathbf{R} + x\mathcal{G} \in \mathbb{Z}_q^{n \times m}$. Output $\mathbf{C}$ as the ciphertext.

- **Decryption:** $x' \leftarrow \mathsf{GSW.Decrypt}(\mathrm{SK}, \mathbf{C})$
  On input a ciphertext $C \in \mathbb{Z}^{n \times m}$ and secret key $\mathrm{SK} = \mathbf{t}$, compute $v = \mathbf{t}\mathbf{C}\mathcal{G}^{-1}(\mathbf{w}^T)$, where $\mathbf{w} = [0, \ldots, \lceil q/2 \rceil] \in \mathbb{Z}^n$. Output $x' = \left| \left\lfloor \frac{v}{q/2} \right\rceil \right|$.

- On input two ciphertexts $\mathbf{C}_1, C_2 \in \mathbb{Z}_q^{n \times m}$, we define homomorphic addition and multiplication:

  - $\mathsf{GSW.Add}(\mathbf{C}_1, \mathbf{C}_2)$: Output $\mathbf{C}_1 + \mathbf{C}_2 \in \mathbb{Z}_q^{n \times m}$.
  - Output the matrix product $\mathbf{C}_1\mathcal{G}^{-1}(\mathbf{C}_2) \in \mathbb{Z}_q^{n \times m}$.

  This allows computation of a NAND gate homomorphically by outputting $\mathcal{G} - \mathbf{C}_1\mathcal{G}^{-1}(\mathbf{C}_2)$.

The following theorem is proved in [32].

Theorem 2.2. The scheme described above is a secure (leveled) FHE scheme under the $\mathsf{LWE}_{n-1,q,\chi,B_\chi}$ assumption.

Note that $\mathbf{tA} = e \approx 0$ since $\mathbf{e}$ is a small error. For correctness, $v = \mathbf{t}(\mathbf{AR} + x\mathcal{G})\mathcal{G}^{-1}(\mathbf{w}^T) \approx x\mathbf{t}\mathbf{w}^T = x\lceil q/2 \rceil$ since $\mathbf{R}$ and $\mathcal{G}^{-1}\mathbf{w}^T$ are composed of $0, 1$ values. Hence by checking whether v is closer to 0 or $\lceil q/2 \rceil$, we can recover $x \in \{0, 1\}$. Let $\mathbf{C}_1$ and $\mathbf{C}_2$ be encryptions of x_1 and x_2 respectively. Then, $\mathbf{C}^+ = \mathbf{C}_1 + \mathbf{C}_2$ is such that $\mathbf{tC}^+ \approx (x_1 + x_2)\mathbf{t}\mathcal{G}$ and $\mathbf{C}^\times = \mathbf{C}_1\mathcal{G}^{-1}(\mathbf{C}_2)$ is such that $\mathbf{tC}^\times \approx (x_1 x_2)\mathbf{t}\mathcal{G}$.

For security, since $\mathbf{A}$ is uniformly random over $\mathbb{Z}_q^{n \times m}$, by leftover hash lemma, for a suitable $m = O(n \log q)$, $\mathbf{AR}$ is statistically uniform, so as $\mathbf{C} = \mathbf{AR} + x\mathcal{G}$.

Key Homomorphic Properties of GSW Scheme. We now show that the GSW scheme satisfies a useful key-homomorphic property, which makes it particularly amendable to convert into a threshold scheme. In particular, we keep the matrix $\mathbf{B}$ fixed, then the sum of two key pairs (computed using $\mathbf{B}$) gives a new valid key pair.

Claim 2.1. *Let* $\mathbf{t}_1 = (-\mathbf{s}_1, 1)$ *and* $\mathbf{t}_2 = (-\mathbf{s}_2, 1)$ *be two secret keys. Let* $\mathbf{B} \in \mathbb{Z}_q^{n-1 \times m}$ *be a random matrix and let* $\mathbf{e}_1$ *and* $\mathbf{e}_2$ *be two error vectors. Further, let* $\mathbf{A}_1 = \begin{bmatrix} \mathbf{B} \\ \mathbf{b}_1 \end{bmatrix} = \mathsf{GSW.PKGen}(\mathbf{t}_1; \mathbf{B}; \mathbf{e}_1)$ *and* $\mathbf{A}_2 = \begin{bmatrix} \mathbf{B} \\ \mathbf{b}_2 \end{bmatrix} = \mathsf{GSW.PKGen}(\mathbf{t}_2; \mathbf{B}; \mathbf{e}_2)$. *Then,* $\mathbf{A} = \begin{bmatrix} \mathbf{B} \\ \mathbf{b}_1 + \mathbf{b}_2 \end{bmatrix} = \mathsf{GSW.PKGen}(\mathbf{t}_1 + \mathbf{t}_2; \mathbf{B}; \mathbf{e}_1 + \mathbf{e}_2)$.

Proof. We have:

$$\mathbf{A}_1 = \mathsf{GSW.PKGen}(\mathbf{t}_1; \mathbf{B}; \mathbf{e}_1) = \begin{bmatrix} \mathbf{B} \\ \mathbf{s}_1\mathbf{B} + \mathbf{e}_1 \end{bmatrix} \tag{2.7}$$

and

$$\mathbf{A}_2 = \mathsf{GSW.PKGen}(\mathbf{t}_2; \mathbf{B}; \mathbf{e}_2) = \begin{bmatrix} \mathbf{B} \\ \mathbf{s}_2\mathbf{B} + \mathbf{e}_2 \end{bmatrix}. \tag{2.8}$$

Hence,

$$\mathbf{A} = \begin{bmatrix} \mathbf{B} \\ \mathbf{b}_1 + \mathbf{b}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{B} \\ (\mathbf{s}_1 + \mathbf{s}_2) \cdot \mathbf{B} + (\mathbf{e}_1 + \mathbf{e}_2) \end{bmatrix} = \mathsf{GSW.PKGen}(\mathbf{t}_1 + \mathbf{t}_2, \mathbf{B}, \mathbf{e}_1 + \mathbf{e}_2) \tag{2.9}$$

QED.

CHAPTER 3: A FRAMEWORK OF BLACK-BOX TRANSFORMATION FOR MPC PROTOCOLS

3.1 CHAPTER 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 [14]. We shall graphically represent a transformation using a circuit diagram like the one in Figure 3.1.

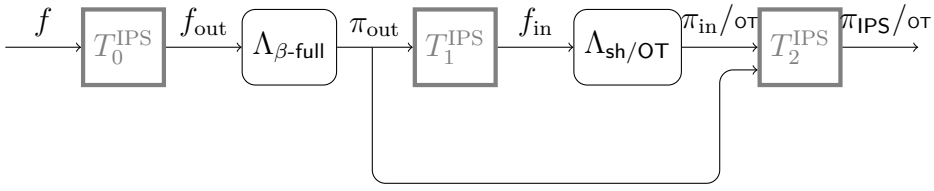

Figure 3.1: Black-Box Transformation in the IPS compiler

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.). [14] 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.

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 [35] 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 $\mathcal{G}$.

3.2 DEFINING BLACK-BOX TRANSFORMATIONS

In this section, we present our framework of black-box transformations, which operates on protocol schemes (Definition 2.2). 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 $f \in \mathcal{F}$,
- 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 $(f, C^{\mathcal{P}_1,\ldots,\mathcal{P}_t}(f)) \in \Lambda$.

3.3 EXAMPLES OF BLACK-BOX TRANSFORMATIONS

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 [18] (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 [15] (from semi-honest secure MPC and and honest-majority secure MPC to active security for Zero-Knowledge proofs) and the IPS compiler [14] (as above, but for arbitrary MPC). The GMW compiler [2] 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 3.1. Similar diagrams for the other examples are given below.

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 $\mathsf{sh\text{-}ppt}$-secure protocol π for f. The second node is a black-box generator, which converts the protocol π to a "zero-knowledge functionality" $f_{ZK}^{R_\pi}$ 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

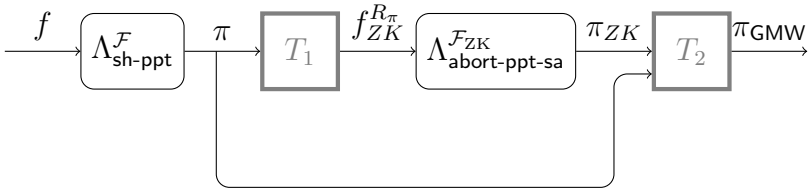

Figure 3.2: BBT from $\{\Lambda_{\text{sh-ppt}}, \Lambda^{\mathcal{F}_{\text{ZK}}}_{\text{abort-ppt-sa}}\}$ to $\Lambda_{\text{abort-ppt-sa}}$ in the GMW compiler

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.

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 [18], originally proposed in the context of Byzantine agreement, and later generalized to MPC protocols (see, e.g., [36]), 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 4.1.)

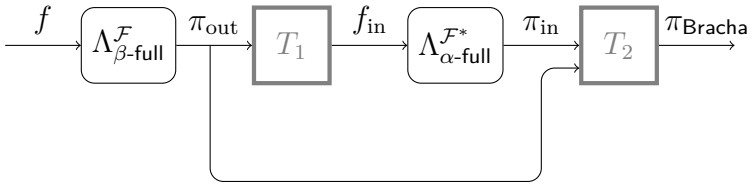

Figure 3.3: Black-Box Transformation in the Bracha compiler

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 [14] The IPS compiler shares a similar structure with the Bracha compiler. Figure 3.1 depicts this transformation. In this figure, T_0^{IPS} converts an n-party functionality into a functionality involving n "clients" and N "servers" (originally $N = O(n^2 k)$ servers in the setting of [14]). 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 [14].)

Finally, the bulk of the compiler is part of the transformation T_2^{IPS}. In Figure 4.1 (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 $\mathcal{W}$,which is in turn implemented using a protocol $w_{\text{IPS}}/\mathsf{OT}$.

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}.

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

IKOS compiler [15]. The IKOS compiler transforms honest-majority MPC protocols to zero-knowledge protocols. It can be illustrated as a BBT circuit, as follows:

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

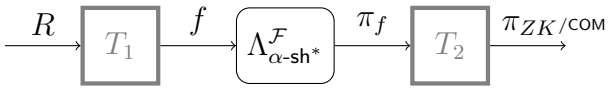

Figure 3.4: A schematic diagram of the IKOS transformation.

of $\Lambda^{\mathcal{F}}_{\alpha\text{-sh}^*}$-scheme compiles f into a semi-honest and *perfectly correct protocol* π_f.[1] 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 [37]. Recently, Hazay and Venkitasubramaniam [37], 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 [37]: our transformation may start with a two-party protocol in the OLE-hybrid model,[2] whereas the one from [37] 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 commitments (to long strings), compared to the protocol in [37] 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 [37] 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.

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

[1] As stated in [15], to improve the efficiency of the whole protocol, this protocol node can be replaced by an actively secure protocol with a (lower) constant threshold.

[2] 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 [38] by generalizing the OT-based GMW protocol [2].

applied to a 1-private protocol) to obtain our full transformation.

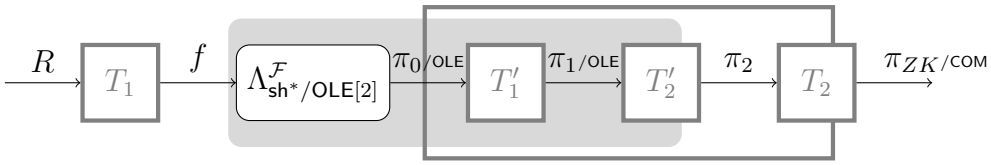

Figure 3.5: A schematic diagram of the new transformation that improves on a result of [37]. T_1 and T_2 are identical to those in Figure 3.4. The shaded components constitue a BBT from 2-party, perfectly correct, semi-honest MPC in the OLE-hybrid to a 3-party, perfectly correct, 1-private MPC in the plain model.

Our transformation, shown in Figure 3.5, follows the IKOS-compiler closely (compare with Figure 3.4 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 3.5). Replacing the 1-private semi-honest protocol in the IKOS BBT of Figure 3.4 with this transformation yields the full BBT in Figure 3.5.

3.3.1 A Pedagogical Application

One of the results from Goldreich's textbook [16] can be simplified using a BBT. In [16], two separate protocols for $\Lambda_{\mathsf{abort\text{-}ppt\text{-}sa\text{-}id}}$ (i.e., security-with-identifiable-abort) and $\Lambda_{1/2-\mathsf{full\text{-}ppt\text{-}sa}}$ (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 $\Lambda_{\mathsf{abort\text{-}ppt\text{-}sa\text{-}id}}$ to $\Lambda_{1/2-\mathsf{full\text{-}ppt\text{-}sa}}$, that uses ECSS (see Section 2.4.2) 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 2.4.2), 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., [39] 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.

3.4 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 [35], 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 3.1. 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 $\Lambda^{\mathcal{F}^*}_{\mathsf{sh}}$.

Proof. 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, $\Pr[(r, x, y, \alpha) \in R] > 1 - \text{negl}(k)$.

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

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 3.2. 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 $\Lambda^{\mathcal{G}}_{\mathsf{abort}}$.

Proof. The functionality family $\mathcal{G}$ that we use to prove Theorem 3.2 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

$$R_O(x, w) = \begin{cases} 1 & \text{if } O(w) = x \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

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:

$$Z^R_{(x',w')}(x, w) = \begin{cases} 1 & \text{if } (x, w) = (x', w') \\ 0 & \text{if } w = w', x \neq x' \\ R(x, w) & \text{otherwise.} \end{cases} \tag{3.2}$$

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 $s \leftarrow \{0,1\}^k$. QED.

CHAPTER 4: NEW BBT CONSTRUCTIONS

4.1 CHAPTER OVERVIEW

New Transformations. We present several new transformations, some of which are summarized in Table 4.1. 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 4.1). 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) [22], which can be realized easily using ordinary Secret-Sharing and one-time message authentication codes (MAC) (see Section 2.5). 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 [14] 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 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

[1]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 4.1, Theorem 4.2	Using player-elimination. Theorem 4.2 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 4.3, Theorem 4.4	An honest-majority version of the IPS transformation. Any $\beta > 0$ suffices. Theorem 4.4 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 4.1	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 4.5	Efficiency Leveraging: resulting protocol almost as efficient as π_1 when there is no abort.[1]
(abort-secure π_1, $t < \alpha n$) and (full-secure π_2, $t < \alpha n$)	full, $t < \alpha n$	Theorem 4.6	Efficiency Leveraging: resulting protocol is almost as efficient as π_1. From Theorem 4.5 and Theorem 4.2. Relies on a non-blackbox decomposition of the function.

Table 4.1: 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.

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).[2]

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

[2]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.

obtain partially-identifiable abort, such that a set of two parties can be identified of which at least one is guaranteed to be corrupt.

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 [40]) 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).

Security with θ-Identifiable Abort. As an intermediate security notion we shall define $\Lambda_{\alpha\text{-}\mathsf{id}}$ which guarantees that on abortion, corruption can be somewhat localized (within two parties, one of whom is guaranteed to be corrupt). The notion generalizes the notion of security with identifiable abort.

Given a normal form functionality f (see Section 2.1), 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 $\mathsf{getoutput}$, 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.

4.2 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 4.5. First, we present a simple but general version of this transformation (which suffices for feasibility results); in Theorem 4.2, we shall present a more efficient variant.

Theorem 4.1. 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 $p\text{-}\Lambda_{\alpha\text{-id/BC}}$ to $(np; \mathsf{D})\text{-}\Lambda_{\alpha\text{-full/BC}}$, 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 2.4.2), which can be realized easily using ordinary Secret-Sharing and one-time message authentication codes (MAC) (see Section 2.5).

Here we give a high level overview of the construction, with a complete description defered to Section 4.6.1. 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 2.3.1 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

$$f = \hat{f}[d] \circ \cdots \circ \hat{f}[1], \tag{4.1}$$

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 Section 4.6.2, we obtain the following efficiency improvement over Theorem 4.1.

Theorem 4.2. For any $0 < \alpha \leq 1/2$, there exists a BBT from $\{\Lambda_{\mathsf{layer}[d]}, \langle \gamma, \delta \rangle \text{-} \Lambda_{\alpha\text{-}\mathsf{id}}\}$ to $(\gamma; \mathsf{D}) \text{-} \Lambda_{\alpha\text{-}\mathsf{full}}$, where $d(n,k) = n \cdot \frac{\delta(n,k)}{\gamma(n,k)}$ and $\mathsf{D}(f) = \mathsf{width}_d(f)$.

4.3 A BBT FROM $\{\Lambda_{\alpha\text{-}\mathsf{SH}}, \Lambda_{\beta\text{-}\mathsf{FULL}}\}$ TO $\Lambda_{\alpha\text{-}\mathsf{ID}}$

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 4.1, 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 4.3. For any $0 < \alpha, \beta \leq 1/2$, there exists a BBT from $\{\Lambda_{\alpha\text{-}\mathsf{sh}}, \Lambda_{\beta\text{-}\mathsf{full}}\}$ to $\Lambda_{\alpha\text{-}\mathsf{id/BC}}$.

The BBT from $(\Lambda_{\beta\text{-}\mathsf{full}}, \Lambda_{\alpha\text{-}\mathsf{sh}})$ to $\Lambda_{\alpha\text{-}\mathsf{id/BC}}$, shown in Figure 4.3(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 $\Lambda_{\alpha\text{-}\mathsf{id/BC}}$ instead of $\Lambda_{\alpha\text{-}\mathsf{abort/OT}}$. Nevertheless, the black-box transformation T_1 is identical to that in IPS, T_1^{IPS}.[3]

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 4.1. We can interpret it as consisting of a "core" compiler $\mathrm{IPS}_{\mathrm{core}}$, 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 Section 4.7.1) was realized using a protocol w_{IPS} in

[3]See Figure 3.1. The functionality f_{in} is a functionality whose trusted party implements the "servers" in the protocol π_{out}.

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 $\mathrm{IPS}_{\mathrm{core}}$ 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 $\widetilde{\mathsf{OT}}$ that is readily realized in the honest-majority setting using a protocol $\pi_{\widetilde{\mathsf{OT}}}$). Similarly, the protocol generated by $\mathrm{IPS}_{\mathrm{core}}$ is modified also to facilitate $1/2$-identification, and to be in the $\mathcal{W}^*$-hybrid than in the $\mathcal{W}$-hybrid. Figure 4.1 shows the new components in our construction, namely the protocol $\pi_{\widetilde{\mathsf{OT}}}$, and two transformations T_{id} and T^*_{id}.

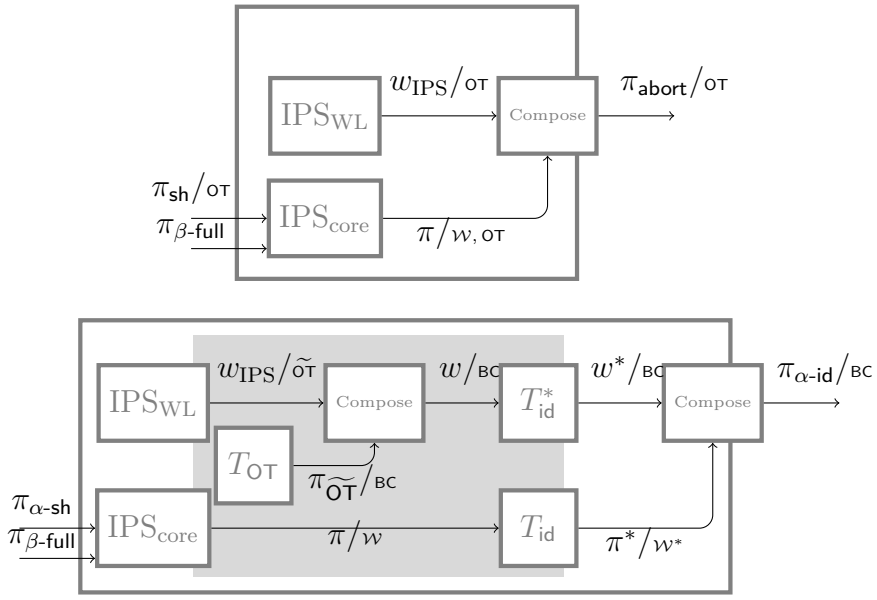

Figure 4.1: T_2^{IPS} and T_2. The shaded region shows the new components in T_2. Note that T_2 retains $\mathrm{IPS}_{\mathrm{core}}$ and $\mathrm{IPS}_{\mathrm{WL}}$ from T_2^{IPS} as it is.

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 $\mathcal{W}^{\langle \mathsf{id}_{1/2} \rangle}$ from the normal form functionality $\mathcal{W}$. $\mathcal{W}^*$ is a weaker functionality obtained by modifying $\mathcal{W}^{\langle \mathsf{id}_{1/2} \rangle}$ 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 $\mathcal{W}^{\langle \mathsf{id}_{1/2} \rangle}$.) A formal description of $\mathcal{W}^*$ is given in Section 4.7.1, where we shall also describe the (simple) transformation T^*_{id} as well as the protocol $\pi_{\widetilde{\mathsf{OT}}}$.

Transformation T_{id}. The goal of the transformation T_{id}, shown in Figure 4.1, 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 4.1 (see paragraph *Obtaining Partially-Identifiable-Abort Security*). A formal description is given in Section 4.7.2.

4.3.1 Using a Sparse Watchlist

The BBT in Theorem 4.3 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 Section 4.7.3). But by exploiting the honest majority guarantee which was absent in the setting of [14], we can state the following version.

Theorem 4.4. For any $0 < \alpha, \beta \leq 1/2$, and polynomials $p_{\text{in}}, q_{\text{in}}, r_{\text{in}}, p_{\text{out}}, q_{\text{out}}$, there exists 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 $p = n \cdot (p_{\text{in}} + r_{\text{in}}) \cdot (q_{\text{out}} + n \cdot p_{\text{out}})$.

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 $(\Lambda_{\beta\text{-full}}, \Lambda_{\alpha\text{-sh}})$ to $\Lambda_{\alpha\text{-id/BC}}$. We present the details in Section 4.8.

4.4 EFFICIENCY LEVERAGING

Bracha's transformation is a classical example of efficiency leveraging. It was originally proposed in the context of byzantine agreement [18], and later applied to MPC protocols (see, e.g., [36]). Below, we record a version of this result that is sufficient for our applications

Proposition 4.1 (Bracha's Transformation [18])**.** *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 $\{(p, q; \mathsf{D})\text{-}\Lambda^{\mathcal{F}}_{\beta\text{-secure}}, p'\text{-}\Lambda_{\alpha\text{-secure}}\}$ to $(p''; \mathsf{D})\text{-}\Lambda^{\mathcal{F}}_{(\alpha-\epsilon)\text{-secure}}$, 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 $\{\Lambda_{\alpha\text{-abort}}, \Lambda_{\alpha\text{-full}}\}$ to $\Lambda_{\alpha\text{-full}}$, in which the resulting protocol's efficiency is comparable to that of the protocol in $\Lambda_{\alpha\text{-abort}}$.

First we present a efficiency leveraging transformation for $\Lambda_{\alpha\text{-id}}$ which can then be combined with Theorem 4.2 to obtain efficiency leveraging for $\Lambda_{\alpha\text{-full}}$. In our efficiency leveraging transformation for $\Lambda_{\alpha\text{-id}}$ the efficiency of the resulting protocol, *when there is*

no abort event, is comparable to that of a cheaper $\Lambda_{\alpha\text{-abort}}$ protocol. Formally, we have the following theorem.

Theorem 4.5. For any $0 \leq \alpha \leq 1/2$, and functions $p, q, p' \in \text{poly}(n, k)$, 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 $\delta = p' \cdot (p + q)$.

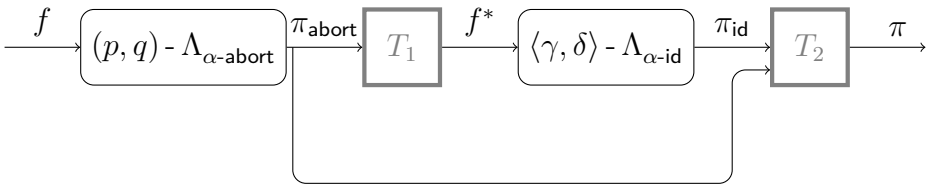

Figure 4.2: Black-Box Transformation from $\{(p, q)\text{-}\Lambda_{\alpha\text{-abort}}, p'\text{-}\Lambda_{\alpha\text{-id}}\}$ to $\langle\gamma, \delta\rangle\text{-}\Lambda_{\alpha\text{-id}}$, where $\gamma = p$ and $\delta = p' \cdot (p + q)$.

The protocol scheme claimed in Theorem 4.5 is shown in Figure 4.2. 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.[4] 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 $p(n, k)$;

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

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 $\langle \gamma, \delta \rangle$ - $\Lambda_{\alpha\text{-}\mathsf{id}}$ with $\gamma = p$ and $\delta' = p' \cdot (p+q)$.

Combining Theorem 4.5 with Theorem 4.2 we get the following result. Here we state it as efficiency leveraging for full-security; however, the result holds as a BBT from $\{\Lambda_{\mathsf{layer}[d]}, (p,q)$ - $\Lambda_{\alpha\text{-}\mathsf{abort}}, p'$ - $\Lambda_{\alpha\text{-}\mathsf{id}}\}$ as well.

Theorem 4.6. 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 $\{\Lambda_{\mathsf{layer}[d]}, (p,q)$ - $\Lambda_{\alpha\text{-}\mathsf{abort}}, p'$ - $\Lambda_{\alpha\text{-}\mathsf{full}}\}$ to $(p; \mathsf{D})$ - $\Lambda_{\alpha\text{-}\mathsf{full}}$, where $d = \frac{n \cdot p' \cdot (p+q)}{p}$ and $\mathsf{D}(f) = \mathsf{width}_d(f)$.

4.5 APPLICATIONS

In Section 3.3.1, 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 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 [11] 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 [9, 10]. Our construction is based on the following direct corollary of Theorem 4.3 and Theorem 4.1.

Corollary 4.1. For any $0 < \alpha, \beta \leq 1/2$, there exists a BBT from $\{\Lambda_{\alpha\text{-}\mathsf{sh}}, \Lambda_{\beta\text{-}\mathsf{full}}\}$ to $\Lambda_{\alpha\text{-}\mathsf{full/BC}}$.

To obtain the result of [11] we simply apply Corollary 4.1 to the protocols in [9, 10].

∘ **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 [41].

Corollary 4.2. 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 4.4[5] and Theorem 4.1. First we obtain a p-$\Lambda_{1/2\text{-id/BC}}$ scheme by applying the BBT from Theorem 4.4 to the $\Lambda_{1/2\text{-sh}}$-scheme from [9] and the constant rate $\Lambda_{\beta\text{-full}}$-scheme (for some $\beta > 0$) that is obtained by instantiating the protocol scheme from [42] using the constant-rate ramp scheme of [43]. (The same "outer protocol" was used in [14] to obtain a constant-rate $\Lambda_{\text{abort/OT}}$-scheme.) Then by further applying the BBT fromTheorem 4.1, 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 4.3. 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 4.5 and Theorem 4.2, by applying the BBTs to the $\Lambda^{\mathcal{F}_{\text{arith}}}_{1/2\text{-abort}}$-scheme from [40] and the p-$\Lambda_{1/2\text{-id}}$-scheme from Corollary 4.2. Note that we have used $\mathsf{width}(f)$ as an upper-bound on $\mathsf{width}_d(f)$ over all d.

Our result complements a similar result of Ben-Sasson et al. [41] 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 $O(n \cdot \mathsf{size}(f) + \text{poly}(n,k))$, compares favorably to the protocols of [41] which yield a cost of $\widetilde{\Omega}(n \cdot \mathsf{size}(f) + n^2 \cdot \mathsf{depth}(f) + \text{poly}(n,k))$.

∘ **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 4.4. 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 $\mathsf{D}(f) = \mathsf{depth}(f)$.

[5]The construction leading to Theorem 4.3 also suffices here. We point to Theorem 4.4 only because it makes the parameters explicit; the optimization in Section 4.3.1 is not important for this result.

This generalizes a result in [36], which obtained a similar result (without using a broadcast channel) for the threshold $\frac{1}{3}-\epsilon$. We obtain this result by applying Proposition 4.1 (Bracha's efficiency leveraging transformation) to our c_n - $\Lambda_{\frac{1}{2}\text{-full/BC}}$ scheme from Corollary 4.2 and the $(c_1, c_2; \mathsf{depth})$ - $\Lambda_{\beta\text{-full}}$ scheme from [36] (for, say, $\beta = 1/6$ and c_1, c_2 being constants), with $\alpha = 1/2$.

4.6 DETAILS OMITTED FROM SECTION 4.2

4.6.1 Details of the Construction in Theorem 4.1

In this section we present the details behind Theorem 4.1. The outline of our BBT is illustrated in Figure 4.3(a).

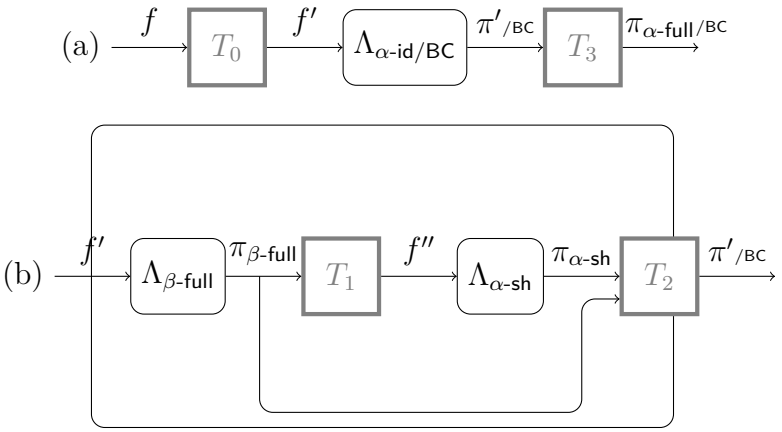

Figure 4.3: (a) BBT from $\{\Lambda_{\alpha\text{-id/BC}}\}$ to $\Lambda_{\alpha\text{-full/BC}}$. (b) BBT from $\{\Lambda_{\beta\text{-full}}, \Lambda_{\alpha\text{-sh}}\}$ to $\Lambda_{\alpha\text{-id}_{\frac{1}{2}}\text{/BC}}$. (a) and (b) together they yield a BBT from $\{\Lambda_{\beta\text{-full}}, \Lambda_{\alpha\text{-sh}}\}$ 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 $\mathsf{reconstruct}$ 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

$$f'(\Sigma_1, \cdots, \Sigma_{n'}) = \mathsf{share}^n_{n'}(f(\mathsf{reconstruct}^n_{n'}(\Sigma_1, \cdots, \Sigma_{n'}))). \tag{4.2}$$

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 of T_3 is as follows.

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 $\pi_{\alpha\text{-id}}[n']$, 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 $\mathsf{reconstruct}_{n'}(\delta^i_1, ..., \delta^i_{n'})$.

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.

4.6.2 Details of the Construction in Theorem 4.2

The new transformation, which is similar to that in Section 4.6.1 is shown in Figure 4.4 (compare with Figure 4.3(a)).

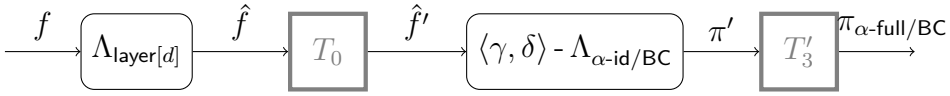

Figure 4.4: A Streamlined version of the BBT in Figure 4.3(a), using layered circuits.

The transformation T_0 here is identical to that in Figure 4.3(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 Section 4.6.1. Below we verify the complexity of this transformation. Let $\hat{f}'[i]$ denote the analog of f' in Equation 4.2. 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 ($e(n,k) = O(n^2k)$ from Section 2.4.2).

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

$$O(\gamma(n,k)\cdot\mathsf{size}(f) + d(n,k)\cdot\gamma(n,k)\cdot e(n,k)\cdot\mathsf{width}_d(f) + \mathrm{poly}(n,k)). \tag{4.3}$$

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

$$O(\gamma(n,k) \cdot \mathsf{size}(f) + \mathrm{poly}(n,k)). \tag{4.4}$$

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

4.7 DETAILS OF THE CONSTRUCTION IN THEOREM 4.3

4.7.1 Watchlist Setup

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 [14] 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 [14], 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. 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 [44]; elementary constructions of AMD codes exist and will suffice for us.[6] Thus, if the adversary alters the shares in any way, *and if* $\beta = x$, then this will

[6] For the sake of completeness, we point out a simple construction $\mathsf{AMDlite}$ due to Cabello et al. [45], that predates the definition of AMD codes: each element $m \in \mathbb{F}$, where $\mathbb{F}$ is an exponentially large field, is

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 $\alpha \leq 1/2$.

Listing of Functionalities and Protocols for Watchlist Setup. We present a slightly modified version of the watchlist functionality used in the IPS compiler [14]. 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 [14], the analysis there allows for this (all servers in the set S are considered to be actively corrupted).

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 $\mathcal{W}$.

encoded as a triplet (m, s, ms), where $s \leftarrow \mathbb{F}$ is uniformly randomly chosen; vectors are encoded by encoding each coordinate independently.

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 $\tilde{L}_i^j \in \{\tilde{\ell} | \mathrm{Pad}_{\tilde{\ell}}^{i \to j} \text{ received}\}$ 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 $\mathrm{Pad}_{(\pi_i^j)^{-1}(\ell)}^{i \to j}$ 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 6) 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.

4.7.2 Formal Description of T_{id}

The components of the transformation T_2 were illustrated in Figure 4.1. 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.

4.7.3 Communication Cost Analysis

Here we analyze the cost of the protocol obtained from the simpler BBT used to prove Theorem 4.3. 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 2.2, 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,[7] and each bit communicated between a pair of parties over the watchlist channel corresponds to $O(n)$ bits

[7]The motivation for the streamlined protocol in Theorem 4.4 is to save this factor of n in setting up the watchlists, by exploiting an honest-majority.

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.

4.8 ANALYSIS FOR THEOREM 4.4

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 [14], 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.

Definition 4.1 ($(n, n'; N, N'; Ph.D.)$-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 $(n, n'; N, N'; Ph.D.)$-Expander if for every $j \in [n]$, we have $|\Gamma(j)| \leq Ph.D.$, 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., [46]), that there are explicit constructions with the following parameters.

Lemma 4.1. 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) $(n, n/2; N, (1-\delta/2)N; Ph.D.)$-expander, where $Ph.D. = c_\delta N/n$.

From the above lemma we can obtain an $(n, n/2; N, (1-\beta/2)N; Ph.D.)$-expander $G_{n,k,\beta}$ where $N = kn^2$ and $Ph.D. = cN/n$ (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 Section 4.8.1.

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.[8] 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 $\pi_{\widetilde{\mathsf{OT}}}$ 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 [14] 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 $(n/2)^2\gamma Ph.D. < \beta N$ 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 $1-\exp(-\Omega(kn))$.

4.8.1 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`

[8] For simplicity, here, as in [14], 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.

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 $\pi_{\alpha\text{-}\mathsf{id}}[n']$, with $\mathcal{P}_j$'s input being $(\sigma^1_j, ..., \sigma^n_j)$ (using $\perp$ for any σ^i_j that was not received).

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 $(\delta^1_j, ..., \delta^n_j)$ and broadcasts "done."

iv. For each $i \in [n]$, on receiving "done" from $n'/2$ or more parties in T^*, $\mathcal{P}_i$ sets $\mathtt{done} = \mathtt{true}$; 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^i_j to $\mathcal{P}_i$.

(d) For each $i \in [n]$, $\mathcal{P}_i$ receives $(\delta^i_1, ..., \delta^j_{n'})$ and sets $x_i = \mathsf{reconstruct}(\delta^i_1, ..., \delta^i_{n'})$.

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 $\mathsf{Pad}^{i \to j}_\ell$ 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 $\{\mathsf{Pad}^{i \to j}_\ell | \ell \in L^j, i \in [n]\{j\}$ 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 $\mathsf{Pad}^{i \to j}_\ell$.

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 $\forall i \in [n], S^j_i = \{\ell | (i, \ell) \in S^j\}$ is of size $|S^j_i| \leq 2\gamma N$. Send $\{\mathsf{Pad}^{i \to j}_i | (i, \ell) \in S^j\}$ to P_j for each corrupted P_j.

- ⋆ 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.

CHAPTER 5: BOTTLENECK COMMUNICATION COMPLEXITY

5.1 CHAPTER OVERVIEW

In this work, we study one fundamental metric of MPC efficiency: the required *communication* between parties. In particular, we focus on the communication complexity of MPC in large-scale settings, where the number of participants is significant.

In nearly all existing works in MPC literature, the communication complexity goal has been to minimize the *total* communication of the protocol across all n parties. However, for many important applications, such as peer-to-peer computations between lightweight devices,[1] total costs (such as total communication) are only secondarily indicative of the feasibility of the computation, as opposed to the primary issue of *per-party* cost. Indeed, while a total communication bound L implies average per-party communication of the protocol is L/n, the computation may demand a subset of the parties to *each* communicate as much as $\Theta(L)$. When all parties contribute input to the computation, then $L \geq n$, meaning these parties must bear communication proportional to the *total number of parties*. In large-scale distributed settings, or when the protocol participants are lightweight devices, such a requirement could be prohibitive.

New efficiency measure: (MPC) Bottleneck Complexity. To address these concerns, we initiate the study of *bottleneck complexity* of MPC. The bottleneck complexity of a protocol Π is defined as the *maximum communication required by any party* within the protocol execution. One may further specialize this to incoming versus outgoing communication. The MPC bottleneck complexity of a (distributed) function is the minimum possible bottleneck complexity of a secure MPC protocol for the function. In this work, our goal is to explore this notion as a complexity measure for distributed computations, and to develop secure protocols with low bottleneck complexity.

Bottleneck complexity addresses certain (practically important) aspects ignored by standard communication complexity. For instance, if two messages are transmitted in two different parts of a network, say $A \to B$ and $C \to D$, they would be delivered faster than two messages sent to/from the same party, say $A \to B$ and $C \to B$. While both have same total communication, the latter has higher bottleneck communication.

Bottleneck Complexity without Security. Before studying bottleneck communication complexity for secure protocols, we first consider this measure for arbitrary protocols without

[1] For example, optimizing navigation routes based on traffic information contributed by the cell phones of drivers on the road, without revealing the locations of individual users.

any security considerations. Indeed, this already forms an interesting measure of complexity for (distributed) functions, and we propose it as a fundamental area to explore. As in the case of total communication complexity (which coincides with bottleneck complexity for the case of 2 parties), there is a trivial upper bound of $O(n)$ bottleneck complexity for any n-party functionality (with boolean inputs), where all parties simply send their input to a central party who computes the functionality. On the other hand, in many functions, bottleneck complexity brings out structures that total communication complexity overlooks. For instance, in computing say the XOR or AND of n bits, total communication complexity is $\Theta(n)$, but the bottleneck complexity is $O(1)$. These functions naturally allow for *incremental computation* along a chain, in which each party receives and sends a single bit. Indeed, there is a large class of useful functions which have protocols with low bottleneck complexity, as discussed below.

However, a priori it is not clear whether *all* functions can be computed in a similar manner. This brings us to the first question considered in this work:

Can all functions be computed (without security)
with *sublinear* bottleneck complexity?

For concreteness, we may consider n-party functions, with n inputs (one for each party) each k bits long, and a single-bit output. Because of the trivial $O(nk)$ upper bound on total communication complexity for any such function (as discussed above), each party *on average* needs only to communicate $O(k)$ bits. But in this protocol, the communication complexity of the central party—and thus bottleneck complexity of the protocol—is $(n-1)k$ bits. Surprisingly, we show that this is the best one can ask for, for general functions. That is, there exist n-party functionalities with k-bit inputs for which the bottleneck complexity is $\Omega(nk)$.

Theorem 5.1. (Informal.) There exist n-party functions with k-bit input for each party that have bottleneck complexity close to that in the trivial upperbound, namely $(n-1)k$.

Our proof is based on a counting argument, and quantifies over possibly inefficient functions too. Interestingly, giving an explicit efficient function f with such a lower bound will require a breakthrough in complexity theory, as it would imply an $\Omega(n^2)$ lower bound on the circuit size of computing f. (We discuss this connection below.)

Functions with Low Bottleneck Complexity. Despite the above lower bound, there is a large class of interesting functions which do have sublinear bottleneck complexity. One simple but widely applicable example is addition in a finite group: the sum of n group elements distributed among n parties can be aggregated bottom-up (and then disseminated top-down)

using a constant-degree tree, with every party communicating $O(d)$ group elements, where d is its degree in the tree.[2]

A wider class of functions are obtained from the literature on streaming algorithms [47, 48]. Indeed, any streaming algorithm with a small memory and a small number of passes corresponds to a low bottleneck complexity function. (Here, we refer to the actual function that the streaming algorithm computes, which may in fact be an approximation to some other desired function.) This is because we can design a protocol which passes around the state of the streaming algorithm from one party to the next, in the order in which their inputs are to be presented to the algorithm.

On the other hand, low bottleneck complexity protocols appear to be much more general than streaming algorithms. Indeed, observe that the low bottleneck complexity protocol described above has a very special communication structure of a chain (or multi-pass cycle). We leave it as an open problem to separate these two notions – i.e., find functions which have low bottleneck complexity protocols, but do not have low-memory streaming algorithms.

Finally, we note that, any n-input function with a constant fan-in circuit of subquadratic size (i.e., $o(n^2)$ gates) has a sublinear bottleneck complexity protocol. To see this, first we note that such a circuit can be made to have constant fan-out as well, by increasing the circuit size by a constant factor.[3] Then, a sublinear bottleneck complexity protocol can be obtained from the circuit by partitioning all the $o(n^2)$ gates roughly equally among the n parties, and letting the parties evaluate the gates assigned to them, communicating with each other when wires cross party boundaries. The communication incurred by each party is bounded by the number of wires incident on all the gates it is assigned, which is $o(n)$.

In Section 5.2, we review the related communication compleixty models and lowerbounds from the literature. In Section 5.3, we introduce a new measure of per-party communication complexity for (distibuted) functions, called bottleneck complexity as well as a semi-malicious security for MPC. In Section 5.4, we demonstrate the existence of n-party functions with k bits of input for each party, that have bottleneck complexity $\Theta(nk)$. Showing an explicit function with $\Omega(n)$ bottleneck complexity will require showing an explicit function with $\Omega(n^2)$ circuit size complexity. On the other hand, we observe that many useful classes of functions do have $o(n)$ bottleneck complexity.

[2]If the group is not abelian, the tree used should be such that its in-order traversal should result in the parties to be ordered in the same way their inputs are ordered in the sum being computed.

[3]Given a gate with fan-out $d > 2$, consider the depth-1 tree T rooted that gate with d leaves being the gates to which its outputs are connected. T can be replaced by an equivalent *binary* tree T' with the same root and leaves, and $d-2$ new internal nodes. The new internal nodes of T' can be "charged" to the leaves of T. On doing this for all gates in the circuit, each gate gets charged at most as many times as its fan-in. Since each gate in the original circuit has constant fan-in, this transformation increases the circuit size by at most a constant factor.

Communication complexity models. The vast majority of study in communication complexity (c.f. [49]) focuses on the setting of only two parties, in which case the total and bottleneck complexities of protocols align (asymptotically). In the multi-party setting, several models are considered regarding how the input to f begins initially distributed among the players. The most common such models are the "number-on-forehead" model, in which parties begin holding all inputs except their own, and the model considered in this work (as is standard in MPC), frequently known as the "number in hand" model, where each party begins with his own input. In all cases, the "communication complexity" within the given model refers to the total communication of all parties.

Communication complexity of MPC. Communication complexity of secure multiparty computation (MPC) has been extensively studied over the years. Communication complexity preserving compilers from insecure to secure protocols were introduced in the 2-party setting by [50]. The setting of MPC with many parties was first predominantly considered in the line of work on scalable MPC [51, 52]. Here the focus was on optimizing the complexity as a function of the circuit size $|C|$, and the resulting n-party protocols have per-party communication $\tilde{O}(|C|/n) + \text{poly}(n)$. Some of these works explicitly achieve load-balancing (e.g., [53, 54]), a goal similar in spirit to bottleneck complexity, where the complexity of the protocol is evenly distributed across the parties. To the best of our knowledge, however, the $\text{poly}(n)$ term in the per-party communication complexity is $\Omega(n)$ in all works aside from [55], which achieves $\tilde{O}(|C|/n)$ *amortized* per-party communication but $\tilde{O}(|C|/n + n)$ *bottleneck* complexity (due to its dependence on [56]).

Communication Locality. A related notion to bottleneck complexity is communication locality [57]. The locality of a party is the number of total other parties it must communicate with throughout the protocol, and the locality of the protocol is the worst case locality of any party. In [57], Boyle et al. studied locality in secure MPC and showed (based on various computational assumptions) that any efficiently computable function has a $\text{polylog}(n)$-locality secure MPC protocol.

Lower bounds on MPC communication complexity. As discussed, lower bounds on standard multi-party communication complexity cannot directly imply meaningful lower bounds on bottleneck complexity, as no such bound can exceed $\Omega(n)$ (attainable by all parties sending their input to a single party), but this implies only a bound of $\Omega(1)$ bottleneck complexity. For *secure* computation, in [58], Damgård et al. showed that securely evaluating a circuit of m multiplication gates requires $\Omega(n^2 m)$ total communication in the information-theoretic security setting. This implies a super-linear lower bound for bottleneck complexity

in their setting. We note, however, that their lower bound does not apply to us, as we consider computational security, and further, their lower bound does not apply to the setting where the number of parties is larger than the security parameter.

5.3 OUR MODEL DEFINITION

5.3.1 Bottleneck Complexity

We introduce a new *per-party* communication metric for distributed computations.

Definition 5.1 (Bottleneck Complexity of Protocol)**.** The *individual communication complexity* of a party P_i in an n-party protocol π, denoted as $\mathcal{CC}_i(\pi)$, is the expected number of bits sent or received by P_i in an execution of π, with worst-case inputs.

The *bottleneck complexity* (BC) of an n-party protocol π is the worst-case communication complexity of any party. That is, $\mathsf{BC}(\pi) = \max_{i\in[n]} \mathcal{CC}_i(\pi)$.

Definition 5.2 (Bottleneck Complexity of Function)**.** The *bottleneck complexity of an n-input function f* is the minimum value of $\mathsf{BC}(\pi)$ when quantified over all n-party distributed protocols π which correctly evaluate f.

Analogously, we define the *MPC bottleneck complexity* of f as the minimum $\mathsf{BC}(\pi)$ quantified over all n-party protocols π which *securely* evaluate f.

Admissible Protocols. We will show techniques that transform general (insecure) protocols to secure ones. Here we define the required minimal assumption of the original protocols, which we refer to as admissibility. Roughly, a protocol is admissible if its next-message function is polynomial-time computable and it has a fixed communication pattern.

Below $\mathbb{Z}_+$ denotes the set of non-negative integers.

Definition 5.3 (Admissible Protocol)**.** Let f be a polynomial function, k be a security parameter, and let $\pi = \{\pi_1, ..., \pi_n\}$ be a possibly randomized n-party protocol, where π_i is a next message function of P_i. Let $x = \{x_1, ..., x_n\}$ and $r = \{r_1, ..., r_n\}$ be the input set and the random string set respectively. Denote $\left\{m^t_{i,j}(x,r)\right\}^{t\in[T]}_{i,j\in[n]}$ as the set of the messages generated by $\pi(x,r)$, and let $|m^t_{i,j}(x,r)| \in [0, f(k)]$ be the length of message from P_i to P_j at time t.[4] We say π is admissible if it satisfies the following two conditions:

- **Polynomial-Time Computable:** For each i, next-message function π_i is expressed by a circuit of fixed polynomial-size in $|x_i| + |r_i|$, with a universally bounded depth.

[4]Precisely, for each $i \in [n], t \in [T]$, $\{m_{i,j}(x,r)\}^t_{j\in[n]} \leftarrow \pi_i\left(x_i, r_i, \{m_{j,i}(x,r)\}^{t\in[t-1]}_{j\in[n]}\right)$

- Fixed Communication Pattern: A protocol π is said to have a *fixed communication pattern* if, irrespective of the input and random-tapes of the parties, the total number of rounds $t_{\max}$ is fixed and there is a function $\text{len} : [t_{\max}] \times [n] \times [n] \to \mathbb{Z}_+$ that maps (t, i, j) to the length of message (possibly 0) from P_i to P_j at round t as determined by π for any view of P_i.

Note that above we allow randomized protocols, as some interesting low bottleneck complexity protocols (e.g., those derived from streaming algorithms) tend to be randomized.

5.3.2 Semi-Malicious Security for MPC

Intuitively, a *semi-malicious* adversary is one who follows the protocol specification (similar to a semi-honest adversary), but who may choose its input and "random" coins for the protocol following any arbitrary PPT strategy. These values may depend (efficiently) on any public setup information such as a CRS or PKI, but must be chosen before the protocol execution begins. Once it has chosen these values, it must follow the protocol as specified, given the chosen input, and using the chosen coins in place of the random coins. We allow the adversary to also abort communication with individual parties at any point in the protocol (which, in our protocols, will invariably result in the honest party aborting).

We remark that a collection of similar but non-identical notions of semi-malicious adversaries have been considered in prior works (e.g., [59, 60]), but with varied requirements on when the adversary must commit to his choice of input/random. We observe that the notion we consider is relatively weak, where all such information is chosen before protocol execution.

More formally, a semi-malicious adversary Adv is modeled as an interactive Turing machine (ITM) which, in addition to the standard tapes, has a special auxiliary tape. At the start of the protocol, A selects for each corrupted party P_i an input x_i and randomness r_i (which may depend on the original inputs of corrupted parties and public setup information), and writes x_i, r_i to its special input auxiliary tape. Adv then honestly follows the protocol specification for the corrupt parties given this input and random tape. At each round, it can also choose any honest party to abort the execution.

We say that a protocol Π evaluating a function f is *semi-malicious secure* if it is a standalone secure protocol, but restricted to PPT semi-malicious adversaries.

5.4 LOWERBOUND ON BOTTLENECK COMPLEXITY OF DISTRIBUTED FUNCTIONS

In this section we show that for most functions f on n inputs (each input could be as short as 1 bit, and the output a single bit delivered to a single party), for any distributed computation protocol π that implements f, the (incoming) bottleneck complexity $\mathsf{BC}(\pi)$ is at least $n - O(\log n)$ bits. In fact, this holds true even without any security requirement. This is tight in the sense that, even with a security requirement, there is a protocol in which only one party has individual communication complexity $\Omega(n)$, and all others have communication proportional to their inputs and outputs (with a multiplicative overhead independent of the number of parties).

This is somewhat surprising since many interesting functions do have protocols with constant communication complexity. As mentioned before, any (possibly randomized) function which has a streaming algorithm or a sub-quadratic sized circuit (with small fan-in gates) gives rise to low botteleneck complexity protocols.

To show our lower-bound, we need to therefore rely on functions with roughly a quadratic lower-bound on circuit size. Given the current lack of explicit examples of such functions, we present an *existential result*, and leave it as a conjecture that there are n-bit input boolean functions with polynomial sized circuits with bottleneck communication complexity of $\tilde{\Omega}(n)$. For simplicity, we discuss the case of perfectly correct protocols, but as we shall point out, a small constant probability of error does not change the result significantly. This result says that there is a function (in fact, most functions) such that the best bottleneck complexity is almost achieved by the trivial protocol, in which one party receives the inputs of all the other $n - 1$ parties and carries out the computation locally.

Theorem 5.2. $\exists f : \{0,1\}^{k \times n} \to \{0,1\}$ such that any n-party, each with k bits input, distributed computation protocol that computes f correctly will have at least one party receiving at least $(n-1)k - O(\log nk)$ bits in the worst-case.

Proof. We establish our lower bound using a counting argument. Consider any (possibly randomized) protocol for computing f. Since we assume perfect correctness, we can fix the random-tapes to yield the smallest bottle-neck communication complexity, and derive a deterministic protocol.

Consider the "input transcript" of a party to include all the messages it received from all the parties, through out the protocol. Given our requirement that the messages in each link is encoded using prefix-free codes, such an input transcript can be parsed into the sequence of messages $m^t_{i,j}$ received by P_j (for all t, j). The same holds for the output transcript for each party.

Now, the behavior of each party P_i is fully specified as a function of its input and its input transcript. That is, the protocol π is completely specified by the set of functions $\{\pi_i\}_{i\in[n]}$ each of which maps an input x_i and an input transcript to an output transcripts (not all such functions may correspond to valid protocols, as they may violate causality and let outgoing messages depend on future incoming messages; but all protocols yield such a set of functions).

Suppose the *incoming* communication foe each party for the protocol is d bits. Then the outgoing communication for each party is at most nd bits. Thus, each π_i can be written as a function from $\{0,1\}^{d+k}$ to $\{0,1\}^{nd}$. There are $2^{2^{d+k}\times nd}$ such functions. Since a protocol is a combination of n such functions, we have:

$$\text{Number of protocols with bottle-neck at most } d \leq 2^{2^{d+k}\times nd\times n}. \tag{5.1}$$

On the other hand, the total number of boolean functions from nk is $2^{2^{nk}}$. Hence,

$$\begin{aligned}
2^{2^{d+k}\times nd\times n} \geq 2^{2^{nk}} &\Rightarrow d+k+\log(n^2 d) \geq nk && (5.2)\\
&\Rightarrow d+k+\log(n^3 k) \geq nk \qquad \text{assuming } d \leq nk && (5.3)\\
&\Rightarrow d \geq (n-1)k - 3\log(nk) && (5.4)
\end{aligned}$$

Thus, if $d \leq nk$, then $d \geq (n-1)k - 3\log(nk)$. That is, $d \geq (n-1)k - O(\log nk)$. QED.

Remark 5.1. If we allow a small constant probability of error, by an averaging argument we can fix the randomness of the parties so that the resulting deterministic protocol will evaluate the function correctly in most of the domain. That is, it evaluates a function whose truth table has a small hamming distance from the original function. The calculation above can be repeated with each protocol accounting for at most $2^{c2^{nk}}$ functions for a constant $c < 1$, leading to $d \geq (n-1)k - O(\log(nk - \log(\frac{1}{1-c})))$.

CHAPTER 6: SECURE TRANSFORMATIONS PRESERVING BOTTLENECK COMPLEXITY

6.1 OVERVIEW

We next turn our attention to achieving low bottleneck complexity for *secure* computation of functionalities. We focus on the general setting where up to $n-1$ out of n parties can be corrupted. As a baseline, we observe that the MPC protocol of Dodis *et al.* [61] based on "additive-spooky encryption" can be easily adapted to obtain generic secure computation with $O(n)$ bottleneck complexity (where $O(n)$ hides factors of the security parameter). Therefore, as in the insecure setting, we focus on constructing MPC protocols with sublinear $o(n)$ bottleneck complexity.

Specifically, we ask the question:

> If a function f can be computed with bottleneck complexity L,
> can it be computed *securely* with the same bottleneck complexity,
> up to a multiplicative overhead of the security parameter?

We note that the goal of sublinear bottleneck complexity is strictly stronger than the recently studied problem of MPC with sublinear communication locality [57]. The locality of a protocol is the maximum number of other parties that any party must communicate with during the course of the protocol. It is easy to see that sublinear bottleneck complexity directly implies sublinear locality (since sending/receiving $o(n)$ bits means that a party can only communicate with $o(n)$ neighbors); however, as locality does not place any requirements on the number of bits communicated by a party, the converse is not true. Indeed, without security requirements, every function has an $O(1)$-local protocol, which is not the case for bottleneck complexity.

We show a general compiler which transforms any (possibly insecure) efficient multi-party protocol Π for computing a function f into a protocol Π' for *securely* computing f, preserving the per-party communication and computation requirements up to $O(\lambda^c)$ factors in the security parameter λ for small constant c. The original protocol Π can have an arbitrary communication pattern; however, we require that this pattern must be fixed a priori (independent of inputs) and known to all parties. Our compiler additionally preserves the topology of communication graph of Π (and in particular, preserves locality).

Theorem 6.1. (Informal.) There is a transformation which maps any (possibly insecure) efficient protocol with fixed-communication-pattern for an n-party distributed function f into

a secure MPC protocol for f with asymptotically (as a function of n) the same communication and computational requirements per party, and using the same communication graph as the original protocol.

The main tools underlying the result include a new notion of *incremental fully homomorphic encryption*, which we show can be instantiated from the Learning With Errors (LWE) assumption via [32], as well as zero-knowledge succinct non-interactive arguments of knowledge (ZK-SNARK) [28] with a "ID-based" simulation-extractability property [62, 63]. We rely on a setup that includes a common random string and a (bare) public-key infrastructure, where all the n parties have deposited keys for themselves, and which all the parties can access for free. The setup can be reused for any number of executions.

6.1.1 Our Techniques

We describe the main ideas underlying our positive result: the bottleneck-complexity-preserving transformation from arbitrary protocols to secure ones.

At a high-level, we follow an intuitive outline for our compiler: (1) We first compile an insecure protocol into a protocol that is secure against semi-honest (or honest-but-curious) adversaries using fully homomorphic encryption (FHE). (2) We then use zero-knowledge succinct arguments of knowledge (ZK-SNARKs) to compile it into a protocol that is (standalone) secure against malicious adversaries. However, we run into several technical challenges along the way, requiring us to develop stronger guarantees for FHE and SNARKs, as well as some other new ideas. We elaborate on these challenges and our solutions below.

Semi-honest Security. A natural starting idea to obtain semi-honest security is to execute an "encrypted" version of the underlying (insecure) protocol by using FHE. Once the parties have the encrypted output, they execute the FHE decryption process to learn the output. The immediate problem with implementing this idea in the multiparty setting is which key must we use for encryption and decryption. If a single party knows the (entire) decryption key, then we cannot guarantee security.

To address this problem, two approaches have been developed in the literature: threshold FHE [60], where the parties jointly generate a public key for an FHE scheme such that each party only knows a share of the decryption key, and multi-key FHE [64], where each party has its own public and secret key pair and FHE evaluation can be performed over ciphertexts computed w.r.t. different public keys.

While these approaches have been shown to suffice for constructing round-efficient MPC

protocols, they are not directly applicable to our setting. This is for two reasons:

- Threshold FHE and multi-key FHE systems are defined in the broadcast model of communication where each party gets to see the messages sent by all the other parties. In contrast, our setting is inherently point-to-point, where a party only communicates with its neighbors in the communication graph of the underlying insecure protocol. Indeed, in order to maintain sublinear bottleneck complexity, we cannot afford each party to communicate with all the other parties.

- Further, in all known solutions for threshold FHE [60] and multi-key FHE [31, 64, 65, 66, 67], the size of one or more protocol messages of each party grows at least linearly with the number of parties. This directly violates our sublinear bottleneck complexity requirement.

To address these issues, we define and implement a new notion of *incremental* FHE (IFHE). Roughly, an IFHE scheme is defined similarly to threshold FHE, with the following key strengthened requirements: a "joint" public key can be computed by incrementally combining shares provided by different parties in an arbitrary order. Similarly, a ciphertext w.r.t. the joint public key can be decrypted by incrementally combining partial decryption shares provided by parties in an arbitrary order. Crucially, the intermediate keys and partial decryption values must be *succinct*.

We construct an IFHE scheme with appropriate security guarantees based on the Gentry-Sahai-Waters FHE scheme [32]. Using IFHE, we are able to directly compile an insecure protocol into a semi-honest secure protocol. In fact, this protocol can withstand a slightly stronger adversary – called a *semi-malicious* adversary [60] – which is allowed to maliciously choose its random tape. This will be crucially exploited in the next step, because without it, one will need to enforce honest random-tapes for all the parties (using n-way coin-tossing-in-the-well) which would incur $\Omega(n)$ communication already.

From Semi-Malicious to Malicious Security. A natural approach to achieve security against malicious adversaries is to use the GMW paradigm [19]. Roughly, in the GMW compiler, each party first commits to its input and random tape. Later, whenever a party sends a message of a semi-malicious protocol,[1] it also proves in ZK to *all* the other parties that it is behaving correctly w.r.t. the committed input and random tape.

The GMW commit-and-prove methodology is problematic in our setting since we cannot allow a party to talk to all other parties (directly or indirectly through the other nodes).

[1]The standard GMW compiler is defined for semi-honest protocols and also involves a coin-tossing step. Here, we consider a natural variant that works for semi-malicious protocols.

Yet, in order to achieve security, each honest party must verify not just that its neighbors behave correctly, but that *all* corrupt parties (many of whom may not directly interact with any honest party) behaved honestly. A priori, these may seem to be contradictory goals.

We address all of these challenges by presenting a new generic compiler for *Verifiable Protocol Execution* (VPE), modeled as a functionality $\mathcal{F}_{\mathsf{vpe}}$. Our protocol Π_{vpe} for implementing $\mathcal{F}_{\mathsf{vpe}}$ asymptotically preserves the per-party communication and computational complexity (up to a multiplicative factor polynomial in the security parameter) of the underlying semi-malicious protocol. We construct Π_{vpe} from two main ingredients: (1) a new commitment protocol that allows the parties to compute a succinct "aggregate" commitment over the inputs and randomness of all of the parties. (2) ZK-SNARKs with a strong extraction property as well as simulation-soundness to ensure that adversary cannot prove false statements even upon receiving simulated proofs. We refer the reader to technical sections for details on our commitment protocol. Here, we discuss our use of ZK-SNARKs.

ID-Based Simulation-Extractable ZK-SNARKs. We rely on ZK-SNARKs to let parties provide not just proofs of correctly computing their own messages, but also of having verified previous proofs *recursively*. This use of SNARKs for recursive verification resembles prior work on proof-carrying data [68, 69]. The key difference is that proof-carrying data only addresses correctness of computation, whereas in our setting, we are also concerned with privacy. In particular, in order to argue security, we also require these proofs to be simulation-sound with extractability (or simply *simulation-extractable*), which presents a significant additional challenge.

The core challenge in constructing simulation-extractable ZK-SNARKs (SE-ZK-SNARKs) arises from the inherent limitation that extraction from the adversary must be non-black-box (since the size of the extracted witness is larger than the proof itself), but the adversary receives simulated proofs which he cannot directly produce on his own. Indeed, for this reason SE-ZK-SNARKs are impossible to achieve with strong universal composability (UC) security [70]. To reduce the security of an SE-ZK-SNARK construction to an underlying knowledge assumption (such as standard SNARKs), one must thus either (a) start with an assumption that guarantees non-black-box extraction even in the presence of an oracle (which can be problematic [71]), or (b) somehow in the reduction be able to provide *the code* to answer the adversary's simulated proof queries, without voiding the reduction by including the simulation trapdoor itself.

Two recent works have presented constructions of SE-ZK-SNARKs, each adopting a different approach. Groth and Maller [72] embody approach (a), constructing full SE-ZK-SNARKs from a new specific pairing-based knowledge assumption which assumes extraction in the presence of black-box access to an oracle with the trapdoor. Alternatively, Garman

et al. [73] take approach (b), basing their construction on standard SNARKs; however, their construction is only applicable to a restricted security model where the statements on which the adversarial prover requests simulated proofs are fixed in advance (in which case these proofs can be hardcoded in the reduction). The case where the adversary's queries are chosen *adaptively* as a function of previously simulated proofs (which we need for our transformation) is not currently addressed in this setting.

We provide a new solution for handling adaptive queries, without relying upon oracle-based assumptions as in [72]. We consider an ID-based notion of SE-ZK-SNARK, where each proof is generated with respect to an identity (chosen from a set of identities that are fixed in advance). In our definition, the adversary must fix a set ID^* of "honest" identities in advance and can then receive simulated proofs on adaptively chosen statements w.r.t. identities from this set. It must then come up with an accepting proof for a new statement w.r.t. an identity $\mathsf{id} \notin \mathsf{ID}^*$.

We show how to transform any SNARK argument system into an ID-based SE-ZK-SNARK by relying on only standard cryptographic assumptions. Very roughly, in our construction, it is possible to "puncture" the trapdoor trap for the CRS w.r.t. an identity set ID^*. A punctured trapdoor $\mathsf{trap}_{\mathsf{ID}^*}$ can only be used to simulate the proofs w.r.t. identities $\mathsf{id} \in \mathsf{ID}^*$, but cannot be used to simulate proofs w.r.t. identities $\mathsf{id} \notin \mathsf{ID}^*$. Using such a punctured trapdoor, we are able to successfully implement approach (b) in the adaptive setting. We implement this idea by using identity-based signatures, which can be readily constructed using certificate chains from a standard signature scheme.

Ultimately, we obtain recursively verifiable ID-based SE-SNARKs generically from signatures and (standard) SNARKs with an "additive extraction overhead." While the latter is a relatively strong requirement, such primitives have been considered in prior work [69, 74] and appears to be as justified as the standard SNARK assumption.

6.2 INCREMENTAL FHE

We define and implement a new notion of incremental FHE (IFHE), which is used within our main positive result. We start by providing syntax and security definitions for IFHE in Section 6.2.1. Next, we describe our construction of IFHE in Section 6.2.2. We refer the readers to Section 2.6.3 for some preliminaries and the Gentry-Sahai-Waters (GSW) FHE scheme [32].

6.2.1 Definitions

(Leveled) Fully Homomorphic Encryption. A fully homomorphic encryption (FHE) scheme consists of algorithms (KEYGEN, ENCRYPT, EVAL, DECRYPT), where (KEYGEN, ENCRYPT, DECRYPT) constitute a semantically secure public-key encryption scheme, and EVAL refers to the homomorphic evaluation algorithm on ciphertexts. An ℓ-leveled FHE scheme supports homomorphic evaluation of circuits of depth at most ℓ.

Incremental FHE. An IFHE scheme is defined similarly to threshold FHE [60], with the following key modifications: a "joint" public key can be computed by incrementally combining shares provided by different parties in an arbitrary order. Similarly, a ciphertext w.r.t. the joint public key can be decrypted by incrementally combining partial decryption shares provided by parties in an arbitrary order. Crucially, the intermediate keys and partial decryption values must be *succinct*.

For technical reasons, it is convenient to describe the joint decryption procedure via three sub-algorithms: A procedure PREDEC which pre-processes a homomorphically evaluated ciphertext to be safe for joint decryption; PARTDEC run by each individual party on a ciphertext (with his share of the secret key) to generate his contribution toward the decryption; and COMBINEDEC which combines the outputs of PARTDEC from each party for a given ciphertext to reconstruct the final decrypted output. In addition to standard semantic security, we also require the output of PARTDEC to hide information about the secret key share that was used; this is captured by the Simulatability of Partial Decryption property below.

We proceed to give a formal definition.

Definition 6.1 (Incremental FHE)**.** An *incremental fully homomorphic encryption (IFHE)* scheme is an FHE scheme with an additional algorithm $\mathsf{IFHE}.\textsc{CombineKeys}$ and with DECRYPT replaced by three algorithms $\mathsf{IFHE}.\textsc{PreDec}$, $\mathsf{IFHE}.\textsc{PartDec}$ and $\mathsf{IFHE}.\textsc{CombineDec}$. By PK_S we denote a combined public key of a subset $S \subseteq [n]$ of parties. Particularly, $\mathrm{PK}_{\{i\}} = \mathrm{PK}_i$ is generated by P_i using the algorithm KEYGEN, and $\mathrm{PK} = \mathrm{PK}_{[n]}$ is the final public key. Similarly, by v_S we denote a combined decryption, and by v_i when $S = \{i\}$. For the completeness of notations, let PK_S and v_S be empty strings when $S = \emptyset$. We describe the syntax of the four algorithms as follows:

- $\mathsf{IFHE}.\textsc{CombineKeys}(\mathrm{PK}_S, \mathrm{PK}_T)$: On input 2 combined public keys $\mathrm{PK}_S, \mathrm{PK}_T$, where $S \cap T = \emptyset$, output a combined public key $\mathrm{PK}_{S \cup T}$.
- $\mathsf{IFHE}.\textsc{PreDec}(\mathrm{PK}, \mathbf{C})$: On input a final public key PK and a ciphertext $\mathbf{C}$, sample a public random $\mathbf{R}$, and output a re-randomized ciphertext $\mathbf{C}'$ of the same plaintext.

- $\mathsf{IFHE}.\textsc{PartDec}(\textsc{pk}, \textsc{sk}_i, \mathbf{C})$: On input a final public key $\textsc{pk}$, ith secret key $\textsc{sk}_i$, ciphertext $\mathbf{C}$, output a partial decryption v_i.

- $\mathsf{IFHE}.\textsc{CombineDec}(v_S, v_T)$: On input 2 partial decryptions v_S, v_T, where $S \cap T = \emptyset$, if $|S \cup T| < n$, output a partial decryption $v_{S \cup T}$; otherwise, output a plaintext y as the final decryption.

Also, we require the following additional properties:

Efficiency: There are polynomials $poly_1(\cdot), poly_2(\cdot)$ such that for any security parameter λ and any $S \subseteq [n], S \neq \emptyset$, $|\textsc{pk}_S| = poly_1(\lambda)$ and $|v_S| = poly_2(\lambda)$.

Correctness: Given a set of plaintexts and a circuit to evaluate, the correctness of IFHE says that the FHE evaluation of the circuit over the ciphertexts can always be decrypted to the correct value, where the ciphertexts are encryption of plaintexts using a single combined public key.

Furthermore, by "Incremental" FHE, we mean that the final combined public key as well as the final combined decryption can be formed in an arbitrary incremental manner. That is, a $\textsc{pk}_i$ can first combine with any other $\textsc{pk}_j$ to form a combined key $\textsc{pk}_{\{i,j\}}$, and $\textsc{pk}_{\{i,j\}}$ can then combine with any other $\textsc{pk}_k$ to form $\textsc{pk}_{\{i,j,k\}}$ or with $\textsc{pk}_{\{k,\ell\}}$ to form $\textsc{pk}_{\{i,j,k,\ell\}}$. The final $\textsc{pk}_{[n]}$ should work for all possible combining orders as long as it collects $\textsc{pk}_1, ..., \textsc{pk}_n$, and it is similar for the combining decryption. We will use a binary tree to describe the combining order and particularly use $tree^0$ to describe the combing public key and $tree^0$ the combing decryption. For $b = 0$ and 1, $tree^b$ contains N^b nodes $\{S_i^b\}_{i \in [N^b]}$, including a root S_1^b. Because the number of leaves equals n, we have $N^b = O(n)$. For brevity, we will name a node by its index, and denote the parent of the i-th node by $\mathsf{parent}(i)$. Also, w.l.o.g., assume each node i has two children, and particularly $\mathsf{parent}(2) = \mathsf{parent}(3) = 1$.

Definition 6.2 (Correctness)**.** For any sequence of plaintexts $x_1, \ldots, x_n$ and circuit f of depth bounded by d, and for all $b = 0, 1$, and for any $S_1^b, ...S_{N^b}^b \in 2^{[n]}$ such that $S_1^b = [n]$, for all $j \in \mathsf{leaves}^b, |S_j^b| = 1$, and for all i, j, k with $i \neq j$ and $k = \mathsf{parent}(i) = \mathsf{parent}(j)$, $S_i^b \cap S_j^b = \emptyset$ and $S_i^b \cup S_j^b = S_k^b$, let EXP.IFHE be the following experiment of an IFHE scheme:

1. $\mathsf{params} \leftarrow \mathsf{IFHE}.\textsc{Setup}(1^\lambda, 1^d)$.
2. $\forall i \in [n], (\textsc{pk}_i, \textsc{sk}_i) \leftarrow \mathsf{IFHE}.\textsc{Keygen}(\mathsf{params})$.

3. $\forall i, j, k \in N^0$ with $i \neq j$ and $k = \mathsf{parent}(i) = \mathsf{parent}(j)$, $\mathrm{PK}_{S_i^0} \leftarrow \mathsf{IFHE}.\textsc{CombineKeys}(\mathrm{PK}_{S_{2i}^0}, \mathrm{PK}_{S_{2i+1}^0})$.
4. $\forall i \in [n], \mathbf{C}_i \leftarrow \mathsf{IFHE}.\textsc{Encrypt}(\mathrm{PK}, x_i)$.
5. $\mathbf{C} \leftarrow \mathsf{IFHE}.\textsc{Eval}(\mathbf{C}_1, \ldots, \mathbf{C}_n, f)$.
6. $C' \leftarrow \mathsf{IFHE}.\textsc{PreDec}(\mathrm{PK}, \mathbf{C})$.
7. $\forall i \in [n], v_i \leftarrow \mathsf{IFHE}.\textsc{PartDec}(\mathrm{PK}, \mathbf{C}', \mathrm{SK}_i)$.
8. $\forall i, j, k \in N^1$ with $i \neq j$ and $\mathsf{parent}(i) = \mathsf{parent}(j) = k \neq 1$, $v_{S_k^1} \leftarrow \mathsf{IFHE}.\textsc{CombineDec}(v_{S_i^1}, v_{S_j^1})$.
9. Output $y \leftarrow \mathsf{IFHE}.\textsc{CombineDec}(v_{S_2^1}, v_{S_3^1})$.

Then the correctness for the scheme holds if and only if

$$\Pr[\mathrm{EXP}.\mathsf{IFHE}(\{x_i\}_{i\in[n]]}; f; \{S_i^b\}_{i\in[N^b]}, b \in \{0,1\}) = f(x_1, ..., x_n)] = 1. \tag{6.1}$$

Semantic security under Combined Keys (against Semi-Malicious Adversary):

Given the parameters prepared in the initial setup, the (corrupted) parties $\{P_j\}_{j\neq i}$, instead of using random strings to compute $\{\mathrm{PK}_i, \mathrm{SK}_i\}_{j\neq i}$, can use an arbitrary string to generate $\{\mathrm{PK}_i, \mathrm{SK}_i\}_{j\neq i}$. Then as long as an honest party generates $(\mathrm{PK}_i, \mathrm{SK}_i)$ independently, the encryption using the final combined public key $(\mathrm{PK}_{[n]}, \mathrm{SK}_{[n]})$ is semantically secure.

Formally, consider the following experiment:

1. $(\mathsf{params}) \leftarrow \textsc{Setup}(1^\lambda, 1^d)$
2. $\forall j \neq i$, Adv computes $(\mathrm{PK}_j, \mathrm{SK}_j)$ according to $\textsc{Keygen}(\mathsf{params})$ but replaces the randomly sampled string by a chosen one. Then Adv computes a combined key $\mathrm{PK}_{[n]\setminus\{i\}}$ according to $\textsc{CombineKeys}$, picks $x \in \{0,1\}$ and sends $(\mathrm{PK}_{[n]\setminus\{i\}}, x)$ to the challenger.
3. The challenger computes $(\mathrm{PK}_i, \mathrm{SK}_i) \leftarrow \textsc{Keygen}(\mathsf{params})$, $\mathrm{PK} \leftarrow \textsc{CombineKeys}(\mathrm{PK}_i, \mathrm{PK}_{[n]\setminus\{i\}})$, and chooses a random bit $\beta \xleftarrow{\$} \{0,1\}$.
   - If $\beta = 0$, it computes $\mathbf{C} = \textsc{Encrypt}(\mathrm{PK}, 0)$.
   - Else, it computes $\mathbf{C} = \textsc{Encrypt}(\mathrm{PK}, x)$.

   And it sends $\mathbf{C}$ to Adv.
4. Finally Adv outputs a bit β'.

We say that the IFHE scheme has semantic security under combined keys if the advantage $\Pr[\beta' = \beta] - 1/2$ is negligible in the security parameter λ.

Simulatability of Partial Decryption: Let $x \in \{0,1\}$ be an input plaintext and $\mathbf{C}'$ be an IFHE encryption of x. There exists a PPT simulator Sim which, given the combined public key pk, ciphertext $\mathbf{C}'$, a plaintext output y, an index $i \in [n]$ and all but the i-th key $\{sk_j\}_{j\in[n]\setminus\{i\}}$, produces a simulated partial decryption v_i' *computationally* close to the honestly generated value v_i:

$$\left\{\mathrm{PK}, \mathbf{C}', v_i'\right\} \overset{c}{\approx} \left\{\mathrm{PK}, \mathbf{C}', v_i\right\}, \tag{6.2}$$

where $v_i \leftarrow \mathsf{IFHE}.\textsc{PartDec}(\mathrm{PK}, \mathrm{SK}_i, \mathbf{C}')$ and $v_i' \leftarrow \textsc{Sim}(\mathrm{PK}, \{\mathrm{SK}_j\}_{j\in[n]\setminus\{i\}}, y, \mathbf{C}')$.

6.2.2 Construction of IFHE

We present an IFHE scheme building on the FHE scheme of Gentry et al. [32]. The Setup, Keygen, Encrypt, and Eval parts are the same as those of [32], while CombineKeys, PreDec, PartDec and CombineDec parts are new. The algorithms are described as follows:

An IFHE scheme.

- **Setup:** $(\mathsf{params}) \leftarrow \mathsf{IFHE}.\textsc{Setup}(1^\lambda, 1^d)$. Same as GSW, where $\mathsf{params} = (q, n, m, \chi, B_\chi, \mathbf{B})$.
- **Key Generation:** $(\mathrm{PK}_i, \mathrm{SK}_i) \leftarrow \mathsf{IFHE}.\textsc{Keygen}(\mathsf{params})$. Same as GSW, where $\mathrm{PK}_i = (\mathbf{B}, \mathbf{b}_i)$ and $\mathrm{SK}_i = \mathbf{t}_i \equiv (-\mathbf{s}_i, 1)$.
- **Combining Keys:** $\mathrm{PK}_{S\cup T} \leftarrow \mathsf{IFHE}.\textsc{CombineKeys}(\mathrm{PK}_S, \mathrm{PK}_T)$
  On input $\mathrm{PK}_S = (\mathbf{B}, \mathbf{b}_S)$ and $\mathrm{PK}_S = (\mathbf{B}, \mathbf{b}_T)$, output $\mathrm{PK}_{S\cup T} = (\mathbf{B}, \mathbf{b}_S + \mathbf{b}_T)$. Particularly $\mathrm{PK}_{[n]}$ is abbreviated as pk or a matrix $\mathbf{A}$.
- **Encryption:** $\mathbf{C}_i \leftarrow \mathsf{IFHE}.\textsc{Encrypt}(\mathrm{PK}, x_i)$. Same as GSW.
- **Evaluation:** $\mathbf{C} \leftarrow \mathsf{IFHE}.\textsc{Eval}(\mathbf{C}_1, ..., \mathbf{C}_\tau; f)$. Same as GSW.
- **Preparing Decryption:** $\mathbf{C}' \leftarrow \mathsf{IFHE}.\textsc{PreDec}(\mathrm{PK}_i, \mathbf{C})$
  On input $\mathrm{PK} = \mathbf{A}$ and $\mathbf{C} \in \mathbb{Z}_q^{n\times m}$, sample a public random matrix $\mathbf{R}$ in $\{0,1\}^{m\times m}$ and output $\mathbf{C}' = \mathbf{C} + \mathbf{AR}$.

- **Partial Decryption:** $v_i \leftarrow \mathsf{IFHE}.\textsc{PartDec}(\textsc{pk}_i, \textsc{sk}_i, \mathbf{C}')$
  On input $\textsc{pk} = \mathbf{A}$, $\textsc{sk}_i \equiv \mathbf{t}_i \equiv (-\mathbf{s}_i, 1)$, and $\mathbf{C}' \in \mathbb{Z}_q^{n \times m}$, sample $\mathbf{e}'_i \leftarrow \chi^m$, set $\mathbf{t}'_i = t_i$ if $i = 1$ and $\mathbf{t}'_i = (-\mathbf{s}_i, 0)$ if $i > 1$, and output $v_i = (\mathbf{t}'_i \mathbf{C}' - \mathbf{e}'_i)\mathcal{G}^{-1}(\mathbf{w}^T)$, where $\mathbf{w} = (0, ..., 0, \lceil q/2 \rceil) \in \mathbb{Z}_q^n$.

- **Combining Decryption:** $\mathsf{IFHE}.\textsc{CombineDec}(v_S, v_T)$
  On input two partial decryptions v_S, v_T with $S \cap T = \emptyset$, compute a partial decryption $v_{S \cup T} = v_S + v_T$. For $|S \cup T| < n$, output $v_{S \cup T}$; for $|S \cup T| = n$, output a plaintext $y = \left\lfloor \frac{v}{q/2} \right\rceil$.

Below we show this scheme fullfils the required properties described in Section 6.2.1.

Lemma 6.1 (Efficiency)**.** For the IFHE scheme, there are polynomial functions $poly_1(\cdot), poly_2(\cdot)$ such that for any security parameter λ and any $S \subseteq [n], S \neq \emptyset$, $|\textsc{pk}_S| = poly_1(\lambda)$ and $|v_S| = poly_2(\lambda)$.

Proof sketch: This is ascribed to the key homomorphism property as in GSW. Specifically, according to the scheme, for all $S, T \subset [n]$, $\textsc{pk}_{S \cup T} = (\mathbf{B}, \mathbf{b}_S + \mathbf{b}_T)$, where $\mathbf{b}_S + \mathbf{b}_T$ is just addition over $\mathbb{Z}_q$, so $|\textsc{pk}_{S \cup T}| = |\textsc{pk}_S| = |\textsc{pk}_T|$. Similarly, $v_{S \cup T} = v_S + v_T$ is also addition over $\mathbb{Z}_q$ so $|v_{S \cup T}| = |v_S| = |v_T|$. Thus, for all $S \subseteq [n], S \neq \emptyset$, $|\textsc{pk}_S| = |\textsc{pk}_1| = poly_1(\lambda)$ and $|v_S| = |v_1| = poly_2(\lambda)$ for some polynomial $poly_1$ and $poly_2$. QED.

Lemma 6.2 (Correctness)**.** The IFHE scheme satisfies the correctness.

Proof sketch: Recall that $\mathbf{A} = \textsc{pk}_{[n]}$, $\mathbf{t}_i = \textsc{sk}_i = (-\mathbf{s}_i, 1)$ and $\mathbf{b}_i = \mathbf{s}_i \mathbf{B} + \mathbf{e}_i$. Let $\mathbf{t} = (-\sum_{i=1}^n \mathbf{s}_i, 1)$. First, for combining keys, because $\textsc{pk}_{S \cup T} = (\mathbf{B}, \mathbf{b}_S + \mathbf{b}_T)$ for all disjoint S and T, we have $\mathbf{A} = (\mathbf{B}, \sum_{i=1}^n \mathbf{b}_{\{i\}})$. Thus, $\mathbf{tA} = \sum_{i=1}^n \mathbf{e}_i \approx 0$.

Second, suppose for now $f(x) = x$ so the FHE evaluation is trivial.

Then in the decryption preparation, we re-randomize the ciphertext with a public random R and so $\mathbf{C}' = \mathbf{C} + \mathbf{AR} = \mathbf{AR}' + x\mathcal{G}$ for some small matrix R'.

Finally for combining decryption, since $v_{S \cup T} = v_S + v_T$, $v_i = (\mathbf{t}'_i \mathbf{C} + \mathbf{e}'_i)\mathcal{G}^{-1}(\mathbf{w}^T)$ and $\sum_{i=1}^n \mathbf{t}'_i = (-\sum_1^n \mathbf{s}_i, 1) = \mathbf{t}$, we have:

$$v_{[n]} + \mathbf{t}'_0 \mathbf{C} \mathcal{G}^{-1}(\mathbf{w}^T) = \left((\sum_{i=1}^n \mathbf{t}'_i)\mathbf{AR}' + x(\sum_{i=1}^n \mathbf{t}'_i)\mathcal{G} + (\sum_{i=1}^n \mathbf{e}'_i) \right) \mathcal{G}^{-1}(\mathbf{w}^T) \approx x\mathbf{t}\mathbf{w}^T = x\lceil q/2 \rceil. \tag{6.3}$$

Hence by checking whether this value is close to 0 or $\lceil q/2 \rceil$, we can recover $x \in \{0, 1\}$ with probability 1.

For general circuit f, since the encryption is the same of GSW, by a similar demonstration of the homomorphism for GSW, the homomorphism property also holds for the IFHE scheme. Therefore putting them together, we have:

$$\Pr[\text{EXP.}\mathsf{IFHE}(\{x_i\}_{i\in[\tau]}; f; \{S_i^b\}_{i\in[2^n-1],b\in\{0,1\}}) = f(x_1, ..., x_\tau)] = 1. \tag{6.4}$$

QED.

Lemma 6.3 (Semantic Security under Combined Keys)**.** The IFHE scheme is secure under combined keys.

Proof sketch: From KEYGEN, the joint distribution $\text{PK}_i = (\mathbf{B}, \mathbf{b}_i)$ is computationally indistinguishable from a uniformly random matrix by the LWE hardness assumption and so is $\text{PK} = A = (\mathbf{B}, \mathbf{b}_i + \mathbf{b}')$ for any $\mathbf{b}' = \sum_{j\neq i} b_j$ which is generated independently of $\mathbf{b}_i$. Consequently $\mathbf{C} \leftarrow \mathsf{GSW.Encrypt}(\text{PK}, x) = \mathbf{AR} + x\mathcal{G}$, where $\mathbf{R} \xleftarrow{\$} \{0,1\}^{m\times m}$. By the leftover hash lemma, $\mathbf{AR}$ is computationally indistinguishable from a uniformly random matrix, and so is $\mathbf{C}$ (Otherwise there is a polynomial-time approach to distinguish $(\mathbf{B}, \mathbf{b}_i)$ from random.) QED.

Lemma 6.4 (Simulatability of Partial Decryption)**.** For the IFHE scheme, given the output y, the ciphertext $\mathbf{C}'$, all but the i-th key $\{\text{SK}_j\}_{j\in[n]\setminus\{i\}}$, the partial decryption y_i is simulatable.

Proof sketch: According to the scheme, after PREDEC, $\mathbf{C}' = \mathbf{C} + \mathbf{AR} = \mathbf{AR}' + y\mathcal{G}$ for some small matrix $\mathbf{R}'$. Recall $\text{PK} = \mathbf{A} = (\mathbf{B}, \mathbf{b})$, where $\mathbf{b} = \sum_{i=1}^n \mathbf{s}_i\mathbf{B} + \mathbf{e}_i$, and $\text{SK}_i = \mathbf{t}_i = (-\mathbf{s}_i, 1)$. Let $\mathbf{t}_0' = (0^{n-1}, 1)$, and recall $t_1' = t_1$ and $t_i' = (-s_i, 0)$ for $i > 1$.

First, for $i > 1$, $v_i = (\mathbf{t}_i'\mathbf{C}' - \mathbf{e}_i')\mathcal{G}^{-1}(\mathbf{w}^T) = -(\mathbf{s}_i\mathbf{BR}' + \mathbf{e}_i')\mathcal{G}^{-1}(\mathbf{w}^T)$, and for $i = 1$, consider $v_1 - \mathbf{t}_0'\mathbf{C}'\mathcal{G}^{-1}(\mathbf{w}^T) = ((\mathbf{t}_1' - \mathbf{t}_0')\mathbf{C}' - \mathbf{e}_1')\mathcal{G}^{-1}(\mathbf{w}^T) = -(\mathbf{s}_1\mathbf{BR}' + \mathbf{e}_1')\mathcal{G}^{-1}(\mathbf{w}^T)$. Note that the distribution $(\mathbf{BR}', v_i) \overset{c}{\cong} (\mathbf{B}', \mathbf{b}'\mathcal{G}^{-1}(\mathbf{w}^T)) \overset{c}{\cong} (\mathbf{BR}', v_1 - \mathbf{t}_0'\mathbf{C}'\mathcal{G}^{-1}(\mathbf{w}^T))$, where $(\mathbf{B}', \mathbf{b}') \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$ by the LWE assumption.

Let $\mathbf{t}_{-1} = (-\sum_{j\neq 1}\mathbf{s}_j, 0)$ and $\mathbf{t}_{-i} = (-\sum_{j\neq i}\mathbf{s}_j, 1)$ for $i \neq 1$. Sample $\tilde{\mathbf{e}} \leftarrow \chi^m$. Then we construct $v_i' = y \cdot \frac{q}{2} - (\mathbf{t}_{-i}\mathbf{C}' + \tilde{\mathbf{e}})\mathcal{G}^{-1}(\mathbf{w}^T)$ for all i.

Note that for $i \neq 1$, $v_i' = y \cdot \frac{q}{2} - (\mathbf{s}_i\mathbf{BR}' + \tilde{\mathbf{e}} + y\mathbf{t}_{-i}\mathcal{G})\mathcal{G}^{-1}(\mathbf{w}^T)$, and for $i = 1$, $v_i' - \mathbf{t}_0'\mathbf{C}'\mathcal{G}^{-1}(\mathbf{w}^T) = y \cdot \frac{q}{2} - (\mathbf{s}_i\mathbf{BR}' + \tilde{\mathbf{e}} + y\mathbf{t}_0'\mathcal{G})\mathcal{G}^{-1}(\mathbf{w}^T)$. Since $y\mathbf{t}_{-i}\mathcal{G}\mathcal{G}^{-1}(\mathbf{w}^T) = y\mathbf{t}_0'\mathbf{w}^T = y \cdot \frac{q}{2}$, we have $v_i' = -(\mathbf{s}_i\mathbf{BR}' + \tilde{\mathbf{e}})\mathcal{G}^{-1}(\mathbf{w}^T)$ for $i \neq 1$ and $v_1' - \mathbf{t}_0'\mathbf{C}'\mathcal{G}^{-1}(\mathbf{w}^T) = -(\mathbf{s}_1\mathbf{BR}' + \tilde{\mathbf{e}})\mathcal{G}^{-1}(\mathbf{w}^T)$. By LWE assumption, the distribution

$$(\mathbf{BR}', v_i') \overset{c}{\cong} (\mathbf{B}'', \mathbf{b}''\mathcal{G}^{-1}(\mathbf{w}^T)) \overset{c}{\cong} (\mathbf{BR}', v_1' - \mathbf{t}_0'\mathbf{C}'\mathcal{G}^{-1}(\mathbf{w}^T)), \tag{6.5}$$

where $(\mathbf{B}'', \mathbf{b}'') \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$.

Thus, for all i, $v_i \stackrel{c}{\cong} v'_i$, and $\left\lfloor \frac{(v_i+(\sum_{j\neq i}\mathbf{t}'_j)\mathbf{C}'\mathcal{G}^{-1}(\mathbf{w}^T)}{q/2} \right\rceil = y = \left\lfloor \frac{(v'_i+(\sum_{j\neq i}\mathbf{t}'_j)\mathbf{C}'\mathcal{G}^{-1}(\mathbf{w}^T)}{q/2} \right\rceil$, together implying $\{\mathbf{A}, \mathbf{C}', \mathbf{R}, v_i\} \stackrel{c}{\cong} \{\mathbf{A}, \mathbf{C}', \mathbf{R}, v'_i\}$.

QED.

Syntax Extension. For ease of use, we extend the syntax of IFHE.JoinKeys for combining several public keys and the syntax of IFHE.CombineDec for combining several partial decryptions at one time.

- **Combining Keys:** $\text{PK}_{S_1\cup,\ldots,\cup S_\ell} \leftarrow \text{IFHE.CombineKeys}(\text{PK}_{S_1}, \ldots, \text{PK}_{S_\ell})$
  On input $\text{PK}_{S_1}, \ldots, \text{PK}_{S_\ell}$, where $S_1, \ldots, S_\ell \subseteq [n]$ are disjoint, compute $\text{PK}_{S_1\cup S_2}$, $\text{PK}_{(S_1\cup S_2)\cup S_3}, \ldots, \text{PK}_{(S_1\cup,\ldots,\cup S_{\ell-1})\cup S_\ell}$ incrementally and finally output $\text{PK}_{S_1\cup,\ldots,\cup S_\ell}$.
  (Alternatively, in our scheme $\text{PK}_{S_1\cup,\ldots,\cup S_\ell} = (\mathbf{B}, \sum_{i=1}^{\ell} \mathbf{b}_{S_i})$.)

- **Combining Decryption:** $v_{S_1\cup,\ldots,\cup S_\ell} \leftarrow \text{IFHE.CombineDec}(v_{S_1}, \ldots, v_{S_\ell})$
  On input partial decryptions $v_{S_1}, \ldots, v_{S_\ell}$, where $S_1, \ldots, S_\ell \subseteq [n]$ are disjoint,, compute a partial decryption $v_{S_1\cup S_2}, v_{(S_1\cup S_2)\cup S_3}, \ldots, v_{(S_1\cup,\ldots\cup S_{\ell-1})\cup S_\ell}$, and finally if $S_1 \cup \ldots \cup S_\ell \subset [n]$ output $v_{S_1\cup\ldots\cup S_\ell}$; otherwise, a plaintext.
  (Alternatively in our scheme $v_{S_1\cup,\ldots\cup S_\ell} = \sum_{i=1}^{\ell} v_{S_i}$, and for $S_1 \cup \ldots \cup S_\ell \subset [n]$, output $v_{S_1\cup\ldots\cup S_\ell}$; otherwise, $y = \left\lfloor \frac{v}{q/2} \right\rceil$.)

6.3 SEMI-MALICIOUS PROTOCOL

In this section, we demonstrate how to convert an arbitrary admissible multi-party distributed protocol Π (as per Definition 5.3) for computing a function f to a protocol Π_{sm} for computing f secure against semi-malicious adversaries, while preserving the *per-party* computation and communication requirements of Π up to $\text{poly}(\lambda)$ multiplicative factors, independent of the number of parties n. In fact, aside from two additional phases where information is communicated along a spanning tree of the communication network induced by Π, our protocol mimics the precise communication patterns of Π.

The communication pattern of the starting protocol Π can be arbitrary, but we require that it be fixed and known a priori (i.e., not data dependent). The same assumption is made for (a bound on) the message length on each active communication channel in each round. We assume without loss of generality that the output of Π is precisely the evaluation of f on parties' inputs and no additional information.

For simplicity of exposition, we present the transformation for deterministic protocols Π; however, as discussed below, our solution can be extended to handle randomized protocols via

a simple coin tossing procedure, leveraging the fact that while the semi-malicious adversary can arbitrarily choose his "randomness," he must commit to these values before the protocol begins.

Let Π be a protocol defined by deterministic next-message function with the following syntax: $(\mu_{i,1}, \ldots, \mu_{i,n}) = \mathsf{NextMsg}(i, t, x_i, \mathsf{Transc}(i, r-1))$, where: i is the relevant party id, t is the present round number, x_i is party i's secret input (including secret randomness), $\mathsf{Transc}(i, t-1)$ denotes the entire transcript held by party i after the previous round $t-1$, and $(\mu_{i,1}, \ldots, \mu_{i,n})$ denote the respective messages to be sent by party i to respective parties $1, \ldots, n$ in this round (where $\mu_{i,j} = \emptyset$ if no message is to be sent from i to j).

Assume a given spanning tree *tree* over the underlying network graph induced by Π. Let $\mathrm{depth}(tree) = d$. Denote P_1 as the party at the root (level 0), $\mathsf{chldrn}(j)$ be the children set of a party P_j, and $\mathsf{parent}(j)$ be the parent of P_j in *tree*. For $i \in [n]$, let $\mathsf{desc}(i)$ denote the set of all descendants of i in the tree.

The protocol Π_{sm} takes place in three phases, as described below.

Semi-Malicious Pattern-Preserving Protocol Π_{sm}

Let the underlying protocol Π be defined by next-message function $\mathsf{NextMsg}$.

Setup

1. $\mathsf{params} \leftarrow \mathsf{IFHE.}\textsc{Setup}(1^\lambda, 1^d)$
2. All parties receive params as a common random string.

Phase 1: Key setup

1. For $\ell = d, \ldots, 0$: For every $i \in [n]$ for which P_i is at level ℓ of *tree*,
   (a) Aggregate public keys:
      i. Denote the received public keys (if any) as $\{\mathrm{PK}_{\mathsf{desc}(j)}\}_{j \in \mathsf{chldrn}(i)}$.
      ii. Generate a IFHE key pair: $(\mathrm{PK}_i, \mathrm{SK}_i) \leftarrow \mathsf{IFHE.}\textsc{Keygen}(\mathsf{params})$.
      iii. Combine keys: $\mathrm{PK}_{\mathsf{desc}(i)} \leftarrow \mathsf{IFHE.}\textsc{CombineKeys}(\mathrm{PK}_i, \{\mathrm{PK}_{\mathsf{desc}(j)}\}_{j \in \mathsf{chldrn}(i)})$.
   (b) Aggregate randomness values (used to rerandomize output ciphertext):
      i. Denote the received random strings (if any) as $\{\mathsf{rand}_j\}_{j \in \mathsf{chldrn}(i)}$.
      ii. Sample a random string: $\mathsf{rand}_i \leftarrow \{0,1\}^\lambda$.
      iii. Combine random values: $\mathsf{rand}_{\mathsf{desc}(i)} = \bigoplus_{j \in \mathsf{chldrn}(i)} \mathsf{rand}_j$.
   (c) If $\ell = 0$ (i.e., root node), let $\mathrm{PK} := \mathrm{PK}_{\mathsf{desc}(i)}$ and $r := \mathsf{rand}_{\mathsf{desc}(i)}$.
   (d) Else, if $\ell \neq 0$, send $\mathrm{PK}_{\mathsf{desc}(i)}$ to parent node $\mathsf{parent}(i)$.
2. For $\ell = 0, \ldots, d-1$: For every $i \in [n]$ for which P_i is at level ℓ of *tree*,

(a) Let PK be the key received from parent $\mathsf{parent}(i)$. Send PK to all children, $\{P_j | j \in \mathsf{chldrn}(i)\}$.

Phase 2: Computation Each party P_i performs the following.

1. Initialize $\widehat{\mathsf{Transc}}(i, 0) \leftarrow \emptyset$.
2. Encrypt input under joint key: $\hat{x}_i \leftarrow \mathsf{IFHE.}\textsc{Encrypt}(\textsc{pk}, x_i)$.
3. For each round $t = 1, \ldots, \mathsf{rounds}$ of the original protocol, do:
   (a) Update transcript: Let $\widehat{\mathsf{Transc}}(i, t) \leftarrow \widehat{\mathsf{Transc}}(i, t-1) \cup \{(j, t, \hat{\mu}_{j,i})\}_{j \in [n]}$, where $\hat{\mu}_{j,i}$ denotes the (encrypted) message sent from P_j to P_i in the previous round (empty if no such message exists).
   (b) Homomorphically evaluate the next-message function: $(\hat{\mu}_{i,1}, \ldots, \hat{\mu}_{i,n}) \leftarrow \mathsf{IFHE.}\textsc{Eval}(\hat{x}_i, \widehat{\mathsf{Transc}}(i, t); \mathsf{NextMsg}(i, t, \cdot, \cdot, \cdot))$.
   (c) For each $j \in [n]$ that P_i sends a message to in this round t
4. Let $\hat{y}$ denote the final evaluated ciphertext held by the root party, corresponding to an encryption of the desired evaluation output.
   Root party rerandomizes using rand: i.e., $\hat{y} \leftarrow \mathsf{IFHE.}\textsc{PreDec}(\textsc{pk}, \mathbf{C}; \mathsf{rand})$.

Phase 3: Decryption

1. For $\ell = 0, \ldots, d-1$: For every $i \in [n]$ for which P_i is at level ℓ of $tree$,
   (a) Let $\hat{y}$ be the ciphertext received from parent $\mathsf{parent}(i)$. Forward $\hat{y}$ to all children, $\{P_j | j \in \mathsf{chldrn}(i)\}$.
2. For $\ell = d, \ldots, 0$: For every $i \in [n]$ for which P_i is at level ℓ of $tree$,
   (a) Denote the received partially decrypted ciphertexts as $\{\hat{y}_{\mathsf{desc}(j)}\}_{j \in \mathsf{chldrn}(i)}$.
   (b) Compute own contribution of decryption: $\hat{y}_i \leftarrow \mathsf{IFHE.}\textsc{PartDec}(\textsc{pk}, \textsc{sk}_i, \hat{y})$.
   (c) Combine decryptions: $\hat{y}_{\mathsf{desc}(i)} \leftarrow \mathsf{IFHE.}\textsc{CombineDec}(\hat{y}_i, \{\hat{y}_{\mathsf{desc}(j)}\}_{j \in \mathsf{chldrn}(i)})$.
   (d) If $\ell \neq 0$ (i.e., not root node), send $\textsc{pk}_{\mathsf{desc}(i)}$ to parent node $\mathsf{parent}(i)$.
3. Root party P_1: Output $y := \hat{y}_{\mathsf{desc}(0)}$.

Theorem 6.2. Let IFHE be an incremental FHE scheme, and Π be an n-party protocol for evaluating a function f with fixed communication pattern. Then the protocol Π_{sm} securely evaluates f against *semi-malicious* corruptions, preserving the per-party computation and communication requirements of Π up to $\mathrm{poly}(\lambda)$ multiplicative factors (independent of the number of parties n). Moreover, the communication pattern of Π_{sm} is identical to that of Π plus two additional traversals of a communication spanning tree of Π.

Remark 6.1 (Handling randomized protocols Π)**.** Our transformation can be modified to support randomized protocols Π while increasing per-party communication (additively) by only $\mathrm{poly}(\lambda)$, by adding the following "coin tossing" procedure. At the conclusion of the key setup phase, each party P_i samples and encrypts a random λ-bit string s_i under the joint IFHE key. These values are incrementally aggregated up to the root of the communication spanning tree to a ciphertext $\hat{s}$ of $s := \bigoplus_{i\in[n]} s_i$, which is then communicated back along the tree to all leaves. In each future round, parties homomorphically evaluate the $\mathsf{NextMsg}$ function of Π, using (encrypted) randomness generated by homomorphically evaluating a pseudo-random function on the (encrypted) seed $\hat{s}$.

We now proceed to prove the Theorem 6.2. The proof takes two main hybrid steps: First, simulating the output computation by relying on the simulatability of partial secryption property of the IFHE. Once this takes place, no knowledge of the honest parties' secret key shares is required. In the next step, we can thus replace the honest parties' inputs with encryptions of 0 by relying on the semantic security of the IFHE under combined keys.

6.3.1 Proof of Security of Semi-Malicious MPC Protocol

In this section, we prove Theorem 6.2.

The communication pattern of Π_{sm} follows by inspection: Phases 1 and 3 each induce a single traversal of the communication spanning tree; Phase 2 directly matches the communication pattern of Π. The per-party computation and communication in Phases 1 and 3 are each $\mathrm{poly}(\lambda)$. The costs in Phase 2 correspond directly to those of Π, except that all computation is performed via homomorphic evaluation, and all communication is sent in encrypted form. Thus both metrics have multiplicative overhead $\mathrm{poly}(\lambda)$.

We proceed to prove semi-malicious security of Π_{sm}. Let Adv be an arbitrary non-uniform polynomial-time *semi-malicious* adversary corrupting a set of parties $M \subseteq \{P_1, \ldots, P_n\}$. We demonstrate the existence of an ideal-world simulator Sim corrupting the same set of parties M, such that for any input vector $\vec{x}$, for any auxiliary input $z \in \{0,1\}^*$, it holds that:

$$\left\{\mathrm{IDEAL}^{f}_{\mathsf{Sim},M}(1^\lambda, \vec{x}, z)\right\}_{\lambda\in N} \approx_c \left\{\mathrm{REAL}^{\Pi}_{\mathsf{Adv},M}(1^\lambda, \vec{x}, z)\right\}_{\lambda\in N}. \tag{6.6}$$

Consider the following simulator $\mathsf{Sim}(1^\lambda, \{x_i\}_{P_i\in M}, y, z)$, where $y = f(x_1, \ldots, x_n)$ is the output received from the ideal functionality.

1. Setup: Sim generates honestly executes IFHE setup.

2. Phase 1 (Key Setup): Sim honestly emulates the actions of honest parties.

3. Phase 2 (Computation): Sim honestly emulates the actions of honest parties, with the following exception: In Step 2, for each honest party $P_i \notin M$, instead of encrypting the (unknown) input and randomness of P_i, Sim instead generates encryptions $\hat{x}_i, \widehat{\mathsf{rand}}_i \leftarrow \mathsf{IFHE.}\textsc{Encrypt}(\textsc{pk}, 0)$ of *zero*.

4. Phase 3 (Decryption): Sim honestly emulates the actions of honest parties, except that the partial decryption values $\hat{y}_i$ of honest parties are instead generated in the following manner. For all but one honest party P_{i^*}, evaluate the partial decryption $\hat{y}_i$ of P_i honestly. For P_{i^*}, do the following. Let $\hat{y}$ denote the final evaluated ciphertext (known to Sim since it is sent to all parties). For each corrupt party P_j, determine his secret key $\textsc{sk}_j$ by reading off the appropriate region of P_j's committed random tape. Then, Sim simulates the partial decryption $\hat{y}_{i^*}$ of P_{i^*} (see Definition 6.1), given the values of $(y, \hat{y}, \{\textsc{sk}_j\}_{j \in [n] \setminus \{i^*\}})$.

We prove indistinguishability of the simulated experiment via the following sequence of hybrids. For each hybrid $\ell \in \{0, 1, 2\}$, denote by $\mathsf{Hyb}^{\ell}_{\mathsf{Adv},M}(1^\lambda, \vec{x}, z)$ the distribution on the inputs and outputs of all parties within the Hybrid ℓ experiment.

Hybrid 0: Real World.

Hybrid 1: (Simulatability of partial decryption.) Same as Hybrid 0, except that the partial decryption values $\hat{y}_i$ of honest parties in Phase 3 are instead generated as done by Sim.

Claim 6.1. *For every non-uniform polynomial-time* Adv*, input vector* $\vec{x}$*, and auxiliary input* z*,* $\mathsf{Hyb}^0_{\mathsf{Adv},M}(1^\lambda, \vec{x}, z) \stackrel{c}{\cong} \mathsf{Hyb}^1_{\mathsf{Adv},M}(1^\lambda, \vec{x}, z)$.

Proof. The values $\hat{y}_i$ for honest parties $i \neq i^*$ are identically distributed. Note that the evaluated ciphertext $\hat{y}$ is *rerandomized* by the value $\mathsf{rand} = \bigoplus_{j \in [n]} \mathsf{rand}_j$. Since Adv is semi-malicious, the choice of rand_j for corrupt parties P_j were made independently of the honestly sampled rand_{i^*}, meaning that rand is uniformly distributed. The claim thus follows directly from IFHE simulatability of partial decryption. QED.

Hybrid 2: (Semantic security under combined keys.) Simulated experiment. Namely, same as Hybrid 1, except that the encryptions generated in Step 2 on behalf of honest parties are replaced by encryptions of zero.

Claim 6.2. *For every non-uniform polynomial-time* Adv*, input vector* $\vec{x}$*, and auxiliary input* z*,* $\mathsf{Hyb}^1_{\mathsf{Adv},M}(1^\lambda, \vec{x}, z) \stackrel{c}{\cong} \mathsf{Hyb}^2_{\mathsf{Adv},M}(1^\lambda, \vec{x}, z)$.

Proof. Note that already in Hybrid 1 all information about the contribution sk_{i^*} of party P_{i^*} to the joint secret key sk is removed. The claim thus follows directly by IFHE semantic security under combined keys. QED.

6.4 FROM SEMI-MALICIOUS TO MALICIOUS SECURITY

In this section we describe how to compile a semi-malicious secure protocol into an actively secure protocol, with the same communication graph. Our compiler relies on common reference string (CRS) and a (bare) public-key setup.

Building Blocks. We use the following cryptographic primitives in our compiler:

- A *simulation-extractable* succinct non-interactive zero-knowledge argument ($\mathsf{SE\text{-}ZK\text{-}SNARK}$) for a polynomial-time computable relation. Unlike recent works such as [72, 73], our notion of $\mathsf{SE\text{-}ZK\text{-}SNARK}$ is identity-based, where each proof is generated w.r.t. an identity.
- A multisignature scheme $(\mathsf{MS.KeyGen}, \mathsf{MS.Sign}, \mathsf{MS.Combine}, \mathsf{MS.MultiVer})$ for implementing the (bare) public-key setup in our construction. In our setting, we can instantiate a multisignature scheme with a $\mathsf{SE\text{-}ZK\text{-}SNARK}$ with additive overhead together with standard signatures.
- A non-interactive perfectly binding commitment scheme COM.
- A family of collision-resistant hash functions $\textsc{HashFamily}$.

We refer the reader to Section 2.6 for the definitions of SNARKs and multisignatures. Below, we present our definition and construction of ID-based $\mathsf{SE\text{-}ZK\text{-}SNARK}$.

6.4.1 ID-Based $\mathsf{SE\text{-}ZK\text{-}SNARK}$

For reasons as discussed in Section 6.1.1, we consider an ID-based notion of $\mathsf{SE\text{-}ZK\text{-}SNARK}$, where each proof is generated with respect to an identity (chosen from a set of identities that are fixed in advance). Proofs for multiple statements can be computed w.r.t. the same identity. Crucially, in our definition of simulation-extractability, the adversary must fix a set ID^* of "honest" identities in advance and can then receive simulated proofs on adaptively chosen statements w.r.t. identities from this set. It must then come up with an accepting proof for a new statement x w.r.t. an identity $\mathsf{id} \notin \mathsf{ID}^*$. We require the existence of a non-black-box extractor who can extract a valid witness for x from such an adversary.

We show how to transform any SNARK argument system into an ID-based SE-ZK-SNARK by relying on only standard cryptographic assumptions. Very roughly, in our construction, it is possible to "puncture" the trapdoor trap for the CRS w.r.t. an identity set ID^*. A punctured trapdoor $\mathsf{trap}_{\mathsf{ID}^*}$ can only be used to simulate the proofs w.r.t. identities $\mathsf{id} \in \mathsf{ID}^*$, but cannot be used to simulate proofs w.r.t. identities $\mathsf{id} \notin \mathsf{ID}^*$. Using such a punctured trapdoor, we are able to construct an extractor while still simulating proofs to the adversary. Specifically, in order to extract from an adversarial prover P^*, we consider an augmented code M that consists of the simulator algorithm $\mathcal{S}$ with a punctured trapdoor $\mathsf{trap}_{\mathsf{ID}^*}$ hardwired in its description, together with the code of prover P^*. Any proof requested by the prover P^* w.r.t. an identity $\mathsf{id} \in \mathsf{ID}^*$ can be computed by $\mathcal{S}$ using the punctured trapdoor. However, since the cheating prover must produce a proof w.r.t. an identity $\mathsf{id} \notin \mathsf{ID}^*$, we can still successfully extract from M by using the SNARK extractor.

Definition. We now present our definition of ID-based SE-ZK-SNARK. Our definition is parametrized w.r.t. an identity set ID. For our application of SE-ZK-SNARK to actively secure MPC, it suffices to work with polynomial-sized identity sets. Below, the prover and verifier algorithms Prove and Verify are extended to receive an identity as an additional input.

Definition 6.3 (ID-Based SE-ZK-SNARK)**.** A SNARK system $(\mathsf{crsGen}, \mathsf{Prove}, \mathsf{Verify})$ for a relation $\mathcal{R}$ with respect to auxiliary input distribution $\mathcal{Z}$ is said to be a SE-ZK-SNARK with respect to identity set ID if there exists a PPT simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ such that the following holds:

- **Computational Zero-Knowledge:** For every $(x, w) \in \mathcal{R}$ and every $\mathsf{id} \in \mathsf{ID}$,

$$(\mathsf{crs}, \mathsf{id}, x, \pi) \overset{c}{\approx} (\mathsf{crs}', \mathsf{id}, x, \pi'), \tag{6.7}$$

  where $\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\lambda)$, $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w, \mathsf{id})$, $(\mathsf{crs}', \mathsf{trap}) \leftarrow \mathcal{S}_1(1^\lambda)$, $\mathsf{trap}_{\mathsf{id}} \leftarrow \mathcal{S}_2(\mathsf{trap}, \mathsf{id})$ and $\pi' \leftarrow \mathcal{S}_3(\mathsf{crs}', \mathsf{id}, \mathsf{trap}_{\mathsf{id}}, x)$.

- **Simulation-Sound Extractability:** There is a negligible function negl such that, for any non-uniform PPT prover P, there exists a polynomial-size extractor $\mathcal{E}_P$, such that $\Pr[\mathsf{out} = 0] \leq \mathrm{negl}(\lambda)$, where the random variable out is the output of the following experiment:

  1. $(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathcal{S}_1(1^\lambda)$
  2. $z \leftarrow \mathcal{Z}$
  3. $\mathsf{ID}^* \leftarrow P(z, \mathsf{crs})$ s.t. $\mathsf{ID}^* \subset \mathsf{ID}$ and $|ID^*| = \mathrm{poly}(\lambda)$.

4. $\mathsf{trap}_{\mathsf{id}^*} \leftarrow \mathcal{S}_2(\mathsf{trap}, \mathsf{id}^*)$, for every $\mathsf{id}^* \in \mathsf{ID}^*$.
5. $(\mathsf{id}, x, \pi) \leftarrow P(z, \mathsf{crs}, \vec{\mathsf{trap}}_{\mathsf{ID}^*})$, where $\vec{\mathsf{trap}}_{\mathsf{ID}^*} = \{\mathsf{trap}_{\mathsf{id}^*}\}_{\mathsf{id}^* \in \mathsf{ID}^*}$.
6. $(\mathsf{id}, x, \pi, w) \leftarrow \mathcal{E}_P(z, \mathsf{crs})$.
7. Output 0 iff:

$$\mathsf{id} \notin \mathsf{ID}^* \wedge \mathsf{Verify}(x, \pi, \mathsf{crs}, \mathsf{id}) = 1 \wedge (x, w) \notin \mathcal{R}. \tag{6.8}$$

Remark 6.2. In our definition of simulation-extractability, we provide the cheating prover P with a special trapdoor $\vec{\mathsf{trap}}_{\mathsf{ID}^*}$ that is "punctured" at the identity set ID^*. Using this special trapdoor, the cheating prover can compute simulated proofs for arbitrary statements w.r.t. any identity $\mathsf{id} \in \mathsf{ID}^*$ on its own. The correctness of a punctured trapdoor is captured in our definition of computational zero-knowledge.

One could alternatively consider a weaker definition of simulation-extractability where instead of giving a punctured trapdoor to the cheating prover, we only provide him oracle access to the simulator algorithm that uses the trapdoor to simulate proofs w.r.t. identities $\mathsf{id} \in \mathsf{ID}^*$ on statements chosen adaptively by the cheating prover. Clearly, Definition 6.3 implies this weaker definition. We choose to work with Definition 6.3 as it enables easier proof of security for our actively secure MPC protocol discussed later in this section.

Construction. We construct an ID-based SE-ZK-SNARK system $(\mathsf{crsGen}, \mathsf{Prove}, \mathsf{Verify})$ for any **NP** language L and identity set ID. We will use the following two ingredients: (1) a witness-indistinguishable SNARK system $(\mathsf{crsGen}', \mathsf{Prove}', \mathsf{Verify}')$ for an **NP** language L' (described below), and (2) an identity-based signature scheme $(\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Sign}, \mathsf{Verify})$ which can be readily constructed using certificate chains from a standard digital signature scheme, which can in turn be based on one-way functions [75].

- $\mathsf{crsGen}(1^\lambda)$: On input a security parameter, compute $\mathsf{crs}' \leftarrow \mathsf{crsGen}'(1^\lambda)$ and $(\mathsf{msk}, \mathsf{mvk}) \leftarrow \mathsf{Setup}(1^\lambda)$. Output $\mathsf{crs} = (\mathsf{crs}', \mathsf{mvk})$.
- $\mathsf{Prove}(\mathsf{crs}, \mathsf{id}, x, w)$: On input a crs $\mathsf{crs} = (\mathsf{crs}', \mathsf{mvk})$, identity id, statement x and witness w, compute a proof $\pi' \leftarrow \mathsf{Prove}'(\mathsf{crs}, x', w)$ for the statement $x' = (x, \mathsf{id}, \mathsf{mvk})$ s.t. $x' \in L'$ iff:
  - $x \in L$, *or*
  - $\exists \sigma$ s.t. $\mathsf{Verify}(\mathsf{mvk}, \mathsf{id}, x, \sigma) = 1$.

  Here, the prover uses the witness w to prove the first part of the statement x'. Output $\pi = \pi'$.

- $\mathsf{Verify}(\mathsf{crs}, \mathsf{id}, x, \pi)$: On input a crs $\mathsf{crs} = (\mathsf{crs}', \mathsf{mvk})$, identity id, statement x and proof π, compute and output $\mathsf{Verify}'(\mathsf{crs}', x', \pi)$ where $x' = (x, \mathsf{id}, \mathsf{mvk})$.

Theorem 6.3 (ID-Based SE-ZK-SNARK)**.** Assuming the existence of one-way functions and a witness-indistinguishable SNARK system for **NP**, the proposed construction is an ID-Based SE-ZK-SNARK for any **NP** relation.

We start by describing the simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$:

- $\mathcal{S}_1(1^\lambda)$: On input a security parameter, compute:
  - $\mathsf{crs}' \leftarrow \mathsf{crsGen}'(1^\lambda)$
  - $(\mathsf{msk}, \mathsf{mvk}) \leftarrow \mathsf{Setup}(1^\lambda)$

  Output $(\mathsf{crs} = \mathsf{crs}', \mathsf{mvk})$ and $\mathsf{trap} = \mathsf{msk}$.

- $\mathcal{S}_2(\mathsf{trap}, \mathsf{id})$: On input a master trapdoor $\mathsf{trap} = \mathsf{msk}$ and identity id, compute and output a "punctured" trapdoor $\mathsf{trap}_{\mathsf{id}} = sk_{\mathsf{id}} \leftarrow \mathsf{Keygen}(\mathsf{msk}, \mathsf{id})$.

- $\mathcal{S}_3(\mathsf{crs}, \mathsf{id}, \mathsf{trap}_{\mathsf{id}}, x)$: On input a crs $\mathsf{crs} = (\mathsf{crs}', \mathsf{mvk})$, identity id, trapdoor $\mathsf{trap}_{\mathsf{id}} = \mathrm{SK}_{\mathsf{id}}$ and statement x, compute:
  - A signature $\sigma \leftarrow \mathsf{Sign}(\mathrm{SK}_{\mathsf{id}}, x)$ on message x using the signing key $\mathrm{SK}_{\mathsf{id}}$ for identity id.
  - A proof $\pi' \leftarrow \mathsf{Prove}'(\mathsf{crs}', x', \sigma)$ for the statement x' (defined as above) using σ as a witness for the second part of x'.

  Output π'.

Now, note that the crs computed by $\mathcal{S}_1$ is identically distributed to the crs computed by the honest algorithm crsGen. Further, for any identity id and statement $x \in L$, the only difference between a proof computed via Prove and $\mathcal{S}_2$ is that the former computes a proof for statement x' (defined as above) using a witness for the first part of x', while $\mathcal{S}_3$ uses a witness for the second part of x'. However, from the witness indistinguishability property of $(\mathsf{crsGen}', \mathsf{Prove}', \mathsf{Verify}')$, it follows that these proofs are computationally indistinguishable.

We next argue the simulation-sound extractability of our construction. For any cheating prover P, our extractor $\mathcal{E}_P$ is the same as defined for the proof system $(\mathsf{crsGen}', \mathsf{Prove}', \mathsf{Verify}')$. From the correctness of $\mathcal{E}_P$, we know that if P outputs an accepting proof π for any statement x w.r.t. identity id, then except with negligible probability, $\mathcal{E}_P$ outputs a

witness w^* s.t. either $(x, w^*) = (x, w) \in \mathcal{R}$ or $w^* = \sigma$ s.t. $\mathsf{Verify}(\mathsf{mvk}, \mathsf{id}, x, \sigma)$. From the security of identity-based signatures, however, the latter case can only happen with negligible probability.

Remark 6.3. In our transformation in the next subsection, we will in fact require the proof system for a *set of* **NP** *relations* $\{R_1, \cdots, R_N\}$, that can be efficiently evaluated given the index of the relation. These relations can be combined into a single "super-relation" $\mathcal{R}$ which consists of $((i, x), w)$ such that $(x, w) \in R_i$, and use a $\mathsf{SE\text{-}ZK\text{-}SNARK}$ system for this relation $\mathcal{R}$. For the sake of readability, the proof and verification for a statement of the form (i, x) with respect to an identity id will be indicated as $\mathsf{Prove}^{\mathsf{id}}_{R_i}(x, w; \mathsf{crs})$ and $\mathsf{Verify}^{\mathsf{id}}_{R_i}(x, \pi, \mathsf{crs})$.

6.4.2 Verifiable Protocol Execution

To abstract out the compiler, we present a "Verifiable Protocol Execution" functionality $\mathcal{F}_{\mathsf{vpe}}$ (parametrized by a protocol ρ which is to be verifiably executed) and present a protocol Π_{vpe} (also parametrized by ρ) that standalone securely realizes $\mathcal{F}_{\mathsf{vpe}}$ against active corruption. Then, in Section 6.4.4 we show that a semi-malicious protocol can be readily turned into a protocol secure against active corruption, using Π_{vpe}.

First, for any semi-malicious protocol ρ, we define the functionality $\mathcal{F}_{\mathsf{vpe}}\langle\rho\rangle$ which accepts the inputs for ρ from all the parties; also it accepts the randomness for the corrupt parties from the adversary, and uniformly samples the randomness for the honest parties. Then it executes the protocol ρ honestly using these inputs and randomness. At each round, if no party issues "abort" it sends the messages generated by the execution to the parties to which the messages are addressed. Formally we define $\mathcal{F}_{\mathsf{vpe}}\langle\rho\rangle$ as follows:

Definition 6.4 (Functionality $\mathcal{F}_{\mathsf{vpe}}$)**.** The functionality $\mathcal{F}_{\mathsf{vpe}}\langle\rho\rangle$, parametrized by a semi-malicious protocol ρ, is defined as follows:

1. For each $i \in [n]$, if P_i is honest, receive input x_i from P_i and sample a (sufficiently long) bit string s_i uniformly at random; else receive (x_i, s_i) from the adversary. Let $\widehat{x}_i = (x_i, s_i)$.

2. Internally run ρ among virtual parties $\tilde{P}_1, \ldots, \tilde{P}_n$, with inputs $\widehat{x}_1, \ldots, \widehat{x}_n$ respectively. At each round τ of this execution carry out the following:

   (a) If P_j is corrupt, and in ρ, $\tilde{P}_i$ sends a message to $\tilde{P}_j$, then forward the message to P_j.

   (b) If the adversary sends (abort, i), send abort to P_i, and also terminate $\tilde{P}_i$ in the internal execution of ρ.

$$\{\text{VK}_i, \text{SK}_i\}_{i\in[n]} \leftarrow \mathsf{MS.KeyGen}(1^\lambda),$$
$$\text{HASH} \leftarrow \text{HASHFAMILY}(1^\lambda),$$
$$\mathsf{crs} \leftarrow \mathsf{SNARK.crsGen}(1^\lambda),$$
$$\mathsf{params}_\rho \leftarrow \rho.\text{SETUP}(1^\lambda, 1^d)$$

Figure 6.1: The params used in the SE-ZK-SNARK system.

(c) If $\tilde{P}_i$ produces an output y_i, send y_i to P_i.

6.4.3 Protocol Π_{vpe}

For any protocol ρ, the protocol $\Pi_{\mathsf{vpe}}\langle\rho\rangle$ implements $\mathcal{F}_{\mathsf{vpe}}\langle\rho\rangle$ (stand-alone) securely against active corruption. As described below, $\Pi_{\mathsf{vpe}}\langle\rho\rangle$ has a setup phase (to create a common reference string), and consists of two phases: an input commitment phase and an execution phase. ρ itself may include a setup phase $\rho.\text{SETUP}$ to create a common reference string. ρ is required to have an *a priori* determined communication pattern, indicating which parties send messages to which parties in each round (and how many rounds T there are in total). Let $\mathsf{in}(i,\tau)$ denote the set of all j such that at round τ, ρ requires P_j to send a message to P_i. We write $\rho(t;(i,j);\{m^\tau_{k,i}\}_{\tau<t,k\in\mathsf{in}(i,\tau)},\widehat{x}_i)$ to denote the message (possibly empty) that P_i should send to P_j at round t, computed from its own input and randomness $\widehat{x}_i$ and the messages $m^\tau_{k,i}$ it received in all previous rounds $\tau < t$. Also, we write $\rho(\mathsf{out};i;\{m^\tau_{k,i}\}_{\tau\le T,k\in\mathsf{in}(i,\tau)})$ to indicate the output to be produced by P_i at the end of the protocol.

Setup. The setup contains four parts: the setup of the PKI environment, A hash function HASH sampled from a hash family $\text{HASHFAMILY}(1^\lambda)$, the common random strings that will be used in the SE-ZK-SNARK system, and the setup of ρ (if any). Formally, $(\mathsf{params}) \leftarrow \Pi_{\mathsf{vpe}}\langle\rho\rangle.\text{SETUP}(1^\lambda, 1^d)$, where params are showed in Figure 6.1

Commitment Phase. The goal of this phase is to construct a short, globally known commitment to the inputs and random-tapes in ρ, for all the parties. This phase uses a (fixed, arbitrary) subgraph T of the communication graph of ρ, which forms a rooted spanning tree.[2] This phase consists of two passes up and down the tree, starting from the leaves. In the first upward pass, the parties essentially use a Merkle-tree to commit to $\widehat{x}_i = (x_i, s_i)$ for all $i \in [n]$ (using a hash function included in the setup). Along with the

[2]We require the communication graph of ρ to be connected and that all parties know this graph (and spanning tree).

hash value, each party also includes a SE-ZK-SNARK proof of correctness in its message to its parent in the tree; correspondingly, each party verifies the the proofs from its children, and also includes this proof in its input to the hash function. Recall that the SE-ZK-SNARK proofs are associated with an identity; all the proofs used in our protocol shall use the prover's index as the identity. The relation proven to one's parent includes the fact that the proofs from the children verified. Though the relations are recursively defined, as we shall see, the complexity of the relations will not grow exponentially with depth, as the relation at a node will depend only on the complexity of *verifying* a SE-ZK-SNARK proof for the relations at its children, and not on computing those relations themselves.

This is followed by a downward pass starting from the root, where the final aggregated commitment α is sent to all the nodes. Here again, the messages are accompanied by appropriate proofs. At the end of this pass, the honest parties are assured that the commitment they have received includes their own input and randomness; however, it is possible that the inputs from the other parties are not correctly included.[3] To address this, we make one more pair of up and down passes, in which a multi-signature of the commitment is created and passed back to all the parties. Verifying this multi-signature assures the honest parties that all the honest parties have received the same commitment.

T is a spanning tree over the set of nodes $[n]$. $\mathsf{chldrn}(j)$ denotes the set of children of j, and $\mathsf{parent}(j)$ is the parent of j (j other than root) in T. W.l.o.g, we assume that the indices of the nodes are sorted such that the root is 1 and for all $j > 1$, $\mathsf{parent}(j) < j$. The party at node j is denoted by P_j.

All messages are assumed to contain a header uniquely identifying its role in the protocol (corresponding to the variable name and indices in the description below).

Commitment Phase of $\Pi_{\mathsf{vpe}}\langle\rho\rangle$. The commitment phase uses SE-ZK-SNARKs for sets of **NP** relations R_j^{up} and R_j^{down} (with j denoting the prover's index for the former and the receiver's for the latter), defined recursively as follows. Here, all the proofs will use the prover's index as the identity.

- $(\phi, w) \in R_j^{\mathsf{up}}$, for $j \in \mathsf{leaves}(T)$, iff $\phi = (\mathsf{crs}, c)$, $w = (\widehat{x}, r)$, $c = \mathsf{COM}(\widehat{x}; r)$, where $\mathsf{leaves}(T)$ denotes the set of leaves of T.
- $(\phi, w) \in R_j^{\mathsf{up}}$, for $j \notin \mathsf{leaves}(T), j \neq 1$ (i.e., not the root), iff $\phi = (\mathsf{crs}, c)$ and $w = (\widehat{x}, \{c_k, \beta_k\}_{k \in \mathsf{chldrn}(j)}, r)$ s.t. $c = \mathsf{COM}(\textsc{Hash}(\{c_k\}_{k \in \mathsf{chldrn}(j)}), \widehat{x}; r))$ and $\forall k \in \mathsf{chldrn}(j)$, $\mathsf{SNARK.Verify}^k_{R_k^{\mathsf{up}}}((\mathsf{crs}, c_k), \beta_k, \mathsf{crs}) = 1$.

[3] One could have hashed signed inputs to assure each honest party that all honest parties' inputs are correctly included in the commitment; however, this leaves room for the adversary to supply the honest parties with commitments which have inconsistent inputs for *the corrupt parties*. The next pass handles both these issues together.

- $(\phi, w) \in R_j^{\mathsf{down}}$, for $j \in \mathsf{chldrn}(1)$ (i.e., children of root), iff $\phi = (\mathsf{crs}, c, \alpha)$, $w = (\widehat{x}, \{c_\ell, \beta_\ell\}_{\ell \in \mathsf{chldrn}(1)}, r)$ s.t. $c = c_j$, $\alpha = \mathsf{COM}(\textsc{Hash}(\{c_\ell\}_{\ell \in \mathsf{chldrn}(1)}), \widehat{x}; r))$, and $\mathsf{SNARK.Verify}^{\ell}_{R_\ell^{\mathsf{up}}}((\mathsf{crs}, c_\ell), \beta_\ell, \mathsf{crs}) = 1$, $\forall \ell \in \mathsf{chldrn}(1) \setminus \{j\}$
- $(\phi, w) \in R_j^{\mathsf{down}}$, for $j \notin \mathsf{chldrn}(1)$, $j \neq 1$, iff $\phi = (\mathsf{crs}, c, \alpha)$, $w = (\widehat{x}, \{c_\ell\}_{\ell \in \mathsf{chldrn}(\mathsf{parent}(j))}, r, c', \gamma)$ s.t. $c = c_j$, $c' = \mathsf{COM}(\textsc{Hash}(\{c_\ell\}_{\ell \in \mathsf{chldrn}(1)}), \widehat{x}; r))$, and $\mathsf{SNARK.Verify}^{\mathsf{parent}(i)}_{R_i^{\mathsf{down}}}((\mathsf{crs}, c', \alpha), \gamma, \mathsf{crs}) = 1$, where $i = \mathsf{parent}(j)$.

Below, for the sake of brevity, instead of specifying $\pi \leftarrow \mathsf{SNARK.Prove}^{\mathsf{id}}_R(\mathsf{crs}, x, w)$, we often leave w implicit and say that π is a proof that $x \in L[R]$ with respect to the identity id. (Here $L[R] = \{x \mid \exists w \text{ s.t. } (x, w) \in R\}$.)

1. **Aggregate Commitments with Proofs.**

   (a) For $j = n$ to 1, P_j proceeds as follows: If j is not a leaf, first it receives (c_k, β_k) from each $k \in \mathsf{chldrn}(j)$ and asserts that $\mathsf{SNARK.Verify}^{k}_{R_k^{\mathsf{up}}}((\mathsf{crs}, c_k), \beta_k, \mathsf{crs}) = 1$. Then, P_j samples r_j and computes $c_j = \mathsf{COM}(\textsc{Hash}(\{c_k\}_{k \in \mathsf{chldrn}(j)}), \widehat{x}_j; r_j)$. Then:

   - P_j, for $j > 1$, sends (c_j, β_j) to $P_{\mathsf{parent}(j)}$, where β_j is a proof that $(\mathsf{crs}, c_j) \in L[R_j^{\mathsf{up}}]$, with associated identity j (prover's index).
   - P_1 sends, $\forall k \in \mathsf{chldrn}(1)$, sends P_k $(\alpha, \sigma_k, \gamma_k)$ where $\alpha = c_1$ where[4] $\sigma_k \leftarrow \mathsf{MS.Sign}(\mathrm{SK}_1, c_k)$, and γ_k is a proof that $(\mathsf{crs}, c_k, \alpha) \in L[R_k^{\mathsf{down}}]$, with identity 1.

2. **Distribute the Aggregated Commitment with Proofs.** Above, children of the root leaf already received α from the root, P_1.

   (a) For $j = 2$ to n, P_j receives $(\alpha, \sigma_j, \gamma_j)$ from $P_{\mathsf{parent}(j)}$, and asserts that $\mathsf{MS.verify}(\mathrm{VK}_{\mathsf{parent}(j)}, c_j, \sigma_j) = 1$, and $\mathsf{SNARK.Verify}^{\mathsf{parent}(j)}_{R_j^{\mathsf{down}}}(\phi, \gamma_j, \mathsf{crs}) = 1$, where $\phi = (\mathsf{crs}, c_j, \alpha)$ if $j \in \mathsf{chldrn}(1)$ and $\phi = (\mathsf{crs}, \alpha)$ otherwise. If the verification succeeds, it sends $(\alpha, \sigma_k, \gamma_k)$ to P_k for each $k \in \mathsf{chldrn}(j)$, where $\sigma_k \leftarrow \mathsf{MS.Sign}(\mathrm{SK}_j, c_k)$, and γ_k is a proof that $(\mathsf{crs}, \alpha) \in L[R_k^{\mathsf{down}}]$, with identity j.

3. **Create the Multisignature.**

   (a) For $j = n$ to 2, P_j sends a combined multisignature $\sigma_j = \mathsf{MS.Combine}(\{\mathrm{VK}_i, \sigma_i\}_{i \in \mathsf{chldrn}(j)} \cup \{\mathrm{VK}_j, \sigma'_j\}, \alpha)$ to $\mathsf{parent}(j)$, where $\sigma'_j = \mathsf{MS.Sign}(\mathrm{SK}_j, \alpha)$.

   (b) P_1 computes a final combined multisignature $\sigma = \mathsf{MS.Combine}(\{\mathrm{VK}_i, \sigma_i\}_{i \in \mathsf{chldrn}(1)} \cup \{\mathrm{VK}_1, \sigma'_1\}, \alpha)$, where $\sigma'_1 = \mathsf{MS.Sign}(\mathrm{SK}_1, \alpha)$.

[4] Here a normal signature, instead of a multi-signature suffices. We overload MS to keep the setup shorter.

4. **Verify the Multisignature.**

   (a) $\forall k \in \mathsf{chldrn}(1)$, P_1 sends σ to P_k.

   (b) For $j = 2$ to n, P_j receives σ from $P_{\mathsf{parent}(j)}$ and checks if $\mathsf{MS.MultiVer}(\{\mathrm{VK}_\ell\}_{\ell\in[n]}, \alpha, \sigma) = 1$. If so, it sends σ to P_k, $\forall k \in \mathsf{chldrn}(j)$; otherwise, P_j aborts.

Execution Phase In each round, each party P_i sends messages to its outgoing neighbors in the round according to the semi-malicious protocol ρ. Meanwhile, similar to the recursive proofs in the commitment phase, P_i also sends a proof showing that 1) P_i computes the message honestly using the input committed in the first phase and the messages it receives so far, 2) it computes an aggregated commitment of its input as well as its children' commits, 3) it verifies the final aggregated commit α 4) it verifies the proofs that the messages received so far all follow ρ honestly.

Therefore, by the (additive-overhead) extraction of the SE-ZK-SNARK, the proofs are recursively (computationally) bound to its previous proofs that according to the computation path of the protocol and finally bound to the input. The input is then bound to the final combined commitment that has been multisigned by all the parties in the first phase. Hence we can argue that, unless the collision resistance of the hash function, the soundness of the commitment scheme, the unforgeability of the multisignature scheme, or the extractionability of the SE-ZK-SNARK scheme fails, the corrupt parties cannot deviate from the protocol without triggering an abort.

The proofs in the protocol are for relations $R^{\rho}_{i,j,t}$ corresponding to proving that a message from P_i to P_j in the t^{th} step is correctly computed according to the input and randomness $\widehat{x}_i$ that P_i committed during the first phase of the protocol and the messages received during the protocol (with proofs, which have been verified by P_i). This is formalized below.

Recall that $\mathsf{in}(i,\tau)$ denotes the set of all j such that at round τ, ρ requires P_j to send a message to P_i. $R^{\rho}_{i,j,t}$ is defined as follows:

• $(\phi, w) \in R^{\rho}_{i,j,t}$ iff $\phi = (\mathsf{crs}, \alpha, m)$, $w = \left(\widehat{x}, h, r, \gamma, c, \sigma, \{m^{\tau}_{k,i}, \theta^{\tau}_{k,i}, \delta^{\tau}_{k,i}\}_{\tau\in[t-1],k\in\mathsf{in}(i,\tau)}\right)$ such that $m = \rho\left(t; (i,j); \{m^{\tau}_{k,i}\}_{\tau\in[t-1],k\in\mathsf{in}(i,\tau)}; \widehat{x}\right)$, $c = \mathsf{COM}(h, \widehat{x}; r)$, and

$$\mathsf{MS.verify}(\mathrm{VK}_{\mathsf{parent}(i)}, c, \sigma) = 1, \tag{6.9}$$

$$\mathsf{MS.verify}(\mathrm{VK}_k, m^{\tau}_{k,i}, \theta^{\tau}_{k,i}) = 1, \tag{6.10}$$

$$\mathsf{SNARK.Verify}^{\mathsf{parent}(i)}_{R^{\mathsf{down}}_i}((\mathsf{crs}, c, \alpha), \gamma, \mathsf{crs}) = 1, \tag{6.11}$$

$$\mathsf{SNARK.Verify}^{k}_{R^{\rho}_{k,i,\tau}}((\mathsf{crs}, \alpha, m^{\tau}_{k,i}), \delta^{\tau}_{k,i}, \mathsf{crs}) = 1, \qquad \forall \tau \in [t-1], k \in \mathsf{in}(i,\tau). \tag{6.12}$$

(Note that the last condition is absent for $t = 1$.)

Execution Phase of $\Pi_{\mathsf{vpe}}\langle\rho\rangle$. For each round t, P_i and P_j, where $i \in \mathsf{in}(j,t)$:

1. P_i computes $m^t_{i,j} = \rho(t;(i,j);\{m^\tau_{k,i}\}_{\tau\in[t-1],k\in\mathsf{in}(i,\tau)},\widehat{x}_i)$, and a proof $\delta^t_{i,j}$ that $m^t_{i,j} \in L[R^\rho_{i,j,t}]$ with respect to the identity i. It also computes a signature $\theta^t_{i,j}$ on $m^t_{i,j}$ using its signing key SK_i. Then it sends $(m^t_{i,j},\theta^t_{i,j},\delta^t_{i,j})$ to P_j.

2. For each $j \in \mathsf{in}(i,t)$, P_i receives $(m^t_{j,i},\delta^t_{j,i})$ from P_j, defines $\phi = (\mathsf{crs},\alpha,m^t_{j,i})$, and asserts that $\mathsf{SNARK.Verify}^j_{R^\rho_{j,i,t}}(\phi,\delta^t_{j,i},\mathsf{crs}) = 1$.

   If this is the last round of the protocol, P_i computes and outputs $y_i = \rho(\mathsf{out},i;\{m^\tau_{k,i}\}_{\tau\in[t-1],k\in\mathsf{in}(i,\tau)},\widehat{x}_i)$.

We prove the security in Section 6.5.

6.4.4 Using VPE To Compile From Semi-Malicious to Malicious Security

If ρ is a semi-malicious secure protocol for a secure function evaluation functionality $\mathcal{F}$, then there is a simple standalone secure protocol for $\mathcal{F}$ in the the $\mathcal{F}_{\mathsf{vpe}}\langle\rho\rangle$-hybrid model: the honest parties will simply send their inputs for $\mathcal{F}$ to $\mathcal{F}_{\mathsf{vpe}}\langle\rho\rangle$, and output the outputs received back (or abort). The security of the protocol follows from the fact that $\mathcal{F}_{\mathsf{vpe}}\langle\rho\rangle$ *ensures* semi-malicious behavior from the adversary in the ρ execution. Indeed, the only actions allowed for the adversary in $\mathcal{F}_{\mathsf{vpe}}\langle\rho\rangle$– choosing an input and randomness for each corrupt party, followed by aborting communication with an honest party at any point – is allowed by the semi-malicious adversary model.

Further, the above compiler continues to be secure even if we replace $\mathcal{F}_{\mathsf{vpe}}\langle\rho\rangle$ with a protocol $\Pi_{\mathsf{vpe}}\langle\rho\rangle$ that *standalone securely realizes* $\mathcal{F}_{\mathsf{vpe}}\langle\rho\rangle$. While typically UC security is required for such composition, note that only a single session of $\mathcal{F}_{\mathsf{vpe}}\langle\rho\rangle$ is invoked above. Formally, a simulator Sim^* for the final protocol in the $\mathcal{F}$ ideal model is constructed as follows: Given an adversary Adv, first we define the (non-black-box) simulator $\mathsf{Sim}_{\mathsf{Adv}}$ for $\Pi_{\mathsf{vpe}}\langle\rho\rangle$ to obtain a hybrid execution in the $\mathcal{F}_{\mathsf{vpe}}\langle\rho\rangle$, with Sim as the adversary. Since $\mathcal{F}_{\mathsf{vpe}}\langle\rho\rangle$ is internally running ρ with virtual parties P_i, we "open it up" and combine the execution of $\{\tilde{P}_i\}_{i\in\mathsf{C}}$ with $\mathsf{Sim}_{\mathsf{Adv}}$ to define a new adversary Adv', which is simply a semi-malicious adversary for ρ. Then we use the semi-malicious security of ρ to obtain the final simulator Sim^*.

6.5 SECURITY OF THE VPE PROTOCOL

The goal is to show that the environment's view (which includes the adversary's view, along with the honest parties' inputs and outputs) in the real world is indistinguisable from its view in the ideal world of $\mathcal{F}_f$, when using an appropriately defined simulator.

To prove the security under the active corruption, we define five hybrids, where the first hybrid denotes the real world execution and the last hybrid denotes the simulator in the ideal world. Let $\widehat{x}_i = (x_i, s_i)$ be the input of P_i and its private random string s_i, and let y_i be the output of P_i at the end of the execution. Every message in $\Pi_{\mathsf{vpe}}\langle\rho\rangle$ from P_i to P_j is of the form $(\mu^t_{i,j}, \pi^t_{i,j})$, where t is the round number, $\pi^t_{i,j}$ is a SE-ZK-SNARK proof about a statement involving crs, $\mu^t_{i,j}$ and possibly α (which is part of a message $\mu^\tau_{j,i}$ for a round τ during the commitment phase). The π component is absent in the messages sent during the multi-signature phase of the commitment phase. Here, the μ components are computed with no reference to the π components in any of the messages.

For each $i = 0, \cdots, 4$, let $\textsc{Hybrid}_i$ be the view of the (stand-alone) environment in the experiments summarized below.

1. $\textsc{Hybrid}_0$ is the environment's view in the execution of $\Pi_{\mathsf{vpe}}\langle\rho\rangle$ in the real world. The environment's view includes the setup, all the messages received by the adversary during the protocol execution, as well as the honest parties' inputs and outputs $\{(x_i, y_i)\}_{i\in\mathsf{H}}$ from this execution.

2. $\textsc{Hybrid}_1$ is generated by the same experiment as $\textsc{Hybrid}_0$ with the following modification: the simulator $(\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3)$ is used to generate the SE-ZK-SNARK scheme's setup crs as well as all the proofs $\pi^t_{i,j}$ given by the honest parties $i \in \mathsf{H}$.

   It directly follows from the zero-knowledge property of SE-ZK-SNARK that $\textsc{Hybrid}_0 \overset{c}{\approx} \textsc{Hybrid}_1$.

3. $\textsc{Hybrid}_2$ is generated by the same experiment as $\textsc{Hybrid}_1$ with the following modification: In the commitment phase, for each $j \in \mathsf{H}$, P_j uses a commitment to a dummy string as c_j. (Note that in this hybrid the proofs are already generated by a simulator; the simulator takes c_j as an input, and does not need the message or randomness used to generate c_j.)

   From the hiding property of COM, we have $\textsc{Hybrid}_1 \overset{c}{\approx} \textsc{Hybrid}_2$.

4. $\textsc{Hybrid}_3$ is in the $\mathcal{F}_{\mathsf{vpe}}\langle\rho\rangle$-hybrid model, with a non-blackbox simulator Sim interacting with the adversary Adv and the functionality $\mathcal{F}_{\mathsf{vpe}}\langle\rho\rangle$. The details of Sim are shown in Figure 6.2. Briefly, it behaves as follows:

- Sim simulates the commitment phase execution of the honest parties in HYBRID_2 faithfully (as it no more depends on their inputs).
- It uses an extractor for $\mathsf{SE\text{-}ZK\text{-}SNARK}$ system to extract the corrupt parties' inputs and randomness for ρ from the commitment phase, and forwards it to $\mathcal{F}_{\mathsf{vpe}}\langle\rho\rangle$.
- It continues with the simulation of the execution phase using the honest parties' messages received from $\mathcal{F}_{\mathsf{vpe}}\langle\rho\rangle$, by adding simulated proofs to them. In this phase Sim simply verifies all the proofs $\pi^t_{i,j}$ sent by each corrupt party P_i to any honest party P_j, and if any verification fails, it sends (abort, j) to $\mathcal{F}_{\mathsf{vpe}}\langle\rho\rangle$.

To complete the proof, we need to argue that $\text{HYBRID}_2 \overset{c}{\approx} \text{HYBRID}_3$.

Extraction from a Tree of Proofs. We describe an efficient procedure to extract witnesses from a "tree of proofs," when the proofs are given using a $\mathsf{SE\text{-}ZK\text{-}SNARK}$ system *with additive polynomial overhead extraction*. Consider a rooted tree Q, with an **NP** relation R_u associated with each node u of the tree such that $(\phi, w) \in R_u$ only if $\phi = (\mathsf{crs}, x)$ and for all $v \in \mathsf{chldrn}(u)$, $\mathsf{Verify}_{R_v}(\phi_v, \pi_v, \mathsf{crs}) = 1$ (additional conditions may be included), where $\phi_v = f_v(\phi, w)$, $\pi_v = g_v(\phi, w)$ for some efficiently computable functions f_v, g_v. Our goal is to extract a consistent set of witnesses w_v for all the nodes v in the tree, from an adversary who gives a proof π_{v_0} for a statement ϕ_{v_0}, where v_0 is the root of the tree, such that (i) if $u = \mathsf{parent}(v)$, then $(\phi_v, w_v) \in R_v$, where, for $v \neq v_0$, $\phi_v = f_v(\phi_u, w_u)$ with $u = \mathsf{parent}(v)$.

We consider an adversary who is given an auxiliary input $z \leftarrow \mathcal{Z}$, a simulated CRS crs, and also a set of trapdoors $\mathsf{trap}_{\mathsf{ID}^*}$ for a set of identities ID^* (for simplicity we consider an a priori fixed set ID^*, which suffices for our purposes). For brevity, we omit identities and $\mathsf{trap}_{\mathsf{ID}^*}$ from the description below, with the understanding that all the proofs are with respect to specific identities (associated with the nodes) that are not contained in ID^*. We shall define a set of machines A_v, E_v for each node in the tree Q as follows. Each A_v will be defined as a PPT machine that will output a statement-proof pair (ϕ_v, π_v) for the relation R_v, and E_v is the extractor guaranteed to exist by Definition 2.5, such that $\text{RT}(E_v) \leq \text{RT}(A_v) + p(\lambda)$.

A_{v_0} is the original adversary Adv which, given inputs (z, crs) and a random tape r^{v_0} outputs (ϕ_{v_0}, π_{v_0}). For all other nodes v in Q we inductively define A_v as follows, in terms of A_u and E_u, where $u = \mathsf{parent}(v)$: It takes an input (z, crs) and a random tape $r^v = r^u || r_v$, (length of r_v to be bounded below), such and runs $E_u(z, \mathsf{crs}, r^u; r_v)$ to obtain (ϕ_u, π_u, w_u), and outputs $(\phi_v, \pi_v) = (f_v(\phi_u, w_u), g_v(\phi_u, w_u))$.

We point out that, thanks to the additive overhead extraction property, A_v remains a polynomial time adversary. Let $q = p + q_0$ be a polynomial where p is as in the definition of

additive overhead extraction, and q_0 is an upperbound on the time needed to compute f_v and g_v for any node in the tree (q is fixed independent of the adversary). Inductively, suppose that for a node u at a depth d, $\mathrm{RT}(A_u) \leq \mathrm{RT}(A_{v_0})+dq(\lambda)$ (base case being $d = 0$ for $u = v_0$). If v is a child of u, then $\mathrm{RT}(A_v) \leq \mathrm{RT}(E_u)+q_0(\lambda) \leq \mathrm{RT}(A_u)+p(\lambda)+q_0(\lambda) \leq \mathrm{RT}(A_{v_0})+(d+1)q(\lambda)$, by the definition of A_v, the additive overhead guarantee, and the inductive assumption and the definition of q. Also, w.l.o.g, we keep the length of the random tapes (for E_u and A_v) upperbounded by the running time of the machines.

Our final extraction procedure consists of E_u hardwired for all nodes u in Q. On input (z, crs, r) it sets $r^{v_0} = r$ (where v_0 is the root of Q), samples r_v of the appropriate lengths for each node v, defines r^v to be the concatenation of r_u for all ancestors u of v in Q, and runs $E_u(z, \mathsf{crs}, r^u; r_v)$ to obtain w_u for each node u. If the proof system used is a $\mathcal{Z}$-auxiliary input $\mathsf{SE\text{-}ZK\text{-}SNARK}$ for the set of relations $\{R_v\}$ for nodes v in a polynomially large relation-tree Q, then by a union bound, the probability that a PPT adversary Adv, given (z, crs) and $\mathsf{trap}_{\mathsf{ID}^*}$ as above, outputs (ϕ, π) that verifies with respect to an $\mathsf{id} \notin \mathsf{ID}^*$, but the above procedure fails to recover a consistent set of witnesses for all nodes is negligible.

$\textsc{Hybrid}_2 \overset{c}{\approx} \textsc{Hybrid}_3$. We briefly sketch the argument that the view of the environment in $\textsc{Hybrid}_3$ is indistinguisable from that in $\textsc{Hybrid}_2$. The main idea is to ensure that if the proofs and signatures sent by a corrupt parties to an honest party during the execution phase verify, then the accompanying message should match what the functionality $\mathcal{F}_{\mathsf{vpe}}\langle\rho\rangle$ would generate and send to the honest party in that round.

To argue this, we shall use the tree extractor from above again on trees of proofs from corrupt parties to honest parties, this time with relations of the form $R^{\rho}_{i,j,t}$ at the root and consisting of nodes of the form $R^{\rho}_{i',j',t'}$ and $R^{\mathsf{down}}_{i'}$ (with $i' \in \mathsf{C}, t' < t$). The extracted witnesses include purported values of $\widehat{x}_{i'}$ and set of messages from honest parties (possibly the empty set) that are "connected to" $m^t_{i,j}$ in ρ (as well as (simulated) proofs accompanying those messages). The messages from the honest parties must exactly be the ones that the simulator forwarded from $\mathcal{F}_{\mathsf{vpe}}\langle\rho\rangle$, by the unforgeability of the signatures.

We shall also argue that the extracted values $\widehat{x}_{i'}$ from the corrupt parties by this extractor must be same ones as extracted by Sim in the commitment step. This relies on unforgeability of signatures used during the commitment phase.

Recall the forest obtained from T by deleting the honest parties' nodes. First consider a corrupt party i' that appears in a tree in that forest which is rooted at $i^* \neq 1$. In this case, the value $\widehat{x}_{i'}$ extracted (from the commitment phase as well as the execution phase) is related to c_{i^*}, the aggregated commitment sent by P_{i^*} to its parent (which is an honest party). Due to the unforgeability of the signature on c_{i^*}, in all the extractions, $\widehat{x}_{i'}$ will be related to the same value of c_{i^*}. Further, due to the binding of COM and the collision-resistance of $\textsc{Hash}$,

Commitment Phase. Sim interacts with the corrupt parties on behalf of the honest parties $\{P_i\}_{i\in\mathsf{H}}$, acccording to the HYBRID_2 experiment. During the commitment phase it simply executes the honest parties in HYBRID_2 faithfully as this execution does not depend on their inputs.

If any of the proofs β_i ($i \in \mathsf{C}$, $\mathsf{parent}(i) \in \mathsf{H}$) or γ_j ($\mathsf{parent}(j) \in \mathsf{C}$, $j \in \mathsf{H}$) fails to verify during the commitment phase, the corresponding simulated honest party aborts and does not contribute to the multi-signature. In this case Sim will send (abort, i) for all $i \in \mathsf{H}$ after the commitment phase is over, and will stop the simulation.

Input Extraction. If the commitment phase completes with all the proofs β_i and γ_j supplied by the adversary being accepted, Sim will try to extract the inputs $\widehat{x}_i$ for all $i \in \mathsf{C}$, using the tree extractor (described earlier), as follows.

Let T be the tree used for aggregating the commitments. At the end of the commitment phase, Sim considers the forest obtained by deleting the set of nodes H from T. Each tree in this forest is denoted as T_i, where i is the root of that tree. For each tree T_i for $i \neq 1$, proceed as follows: redefine the output of the adversary to be just the proof and the statement for the relation R_i^{up} (i.e., $((\mathsf{crs}, c_i), \beta_i)$), and the auxiliary input Z_i to be all the messages sent to the corrupt parties by Sim (on behalf of honest parties) prior to that, as well as the common reference string of $\Pi_{\mathsf{vpe}}\langle\rho\rangle$; define the relation-tree with R_j^{up} for all j in T_i, rooted at R_i^{up}, and invoke the tree extractor for this relation tree and this adversary, with auxiliary input Z_i. If the extraction succeeds, it yields witnesses for all the proofs β_j for all j in T_i, and in particular, $\widehat{x}_j$ for all parties j in T_i. If P_1 is corrupt we define an adversary which outputs a statement $(\mathsf{crs}, c_k, \alpha)$ and a proof γ_k for the relation R_k^{down}, where P_k is an (arbitrary) honest party adjacent to a leaf of T_1. We define a relation-tree as follows: it consists of R_j^{down} for all j in the path from k to 1 (the root) and R_j^{up} for all $j \in T_1$ (except $j = 1$); this tree is rooted at R_k^{down}. Then we invoke the tree-extractor for the adversary with respect to this relation-tree, with auxiliary input Z_1 which includes all the messages Sim sent to it till it produced the output. If successful, the extraction obtains the witnesses for all the proofs γ_j for all j in the path from $\mathsf{parent}(k)$ till the root, and for all the proofs β_j for all j in T_1 (except $j = 1$). The witness for γ_j for $j \in \mathsf{chldrn}(1)$ includes $\widehat{x}_1$ and the witnesses for β_j include $\widehat{x}_j$ for all other j in T_1.

If all the extractions are successful, Sim forwards the inputs $\widehat{x}_j$ for all corrupt parties P_j obtained above are forwarded to $\mathcal{F}_{\mathsf{vpe}}\langle\rho\rangle$.

Execution Phase. Sim simulates the execution phase using the honest parties' messages received from $\mathcal{F}_{\mathsf{vpe}}\langle\rho\rangle$ at each round, by adding simulated proofs to them. Also Sim faithfully verifies all the signatures (c_i, σ_i) and proofs $\delta_{i,j}^t$ sent by each corrupt party P_i to any honest party P_j, and if any verification fails, it sends (abort, j) to $\mathcal{F}_{\mathsf{vpe}}\langle\rho\rangle$.

Figure 6.2: Simulator Sim used in the HYBRID_3 (ideal execution of $\mathcal{F}_{\mathsf{vpe}}\langle\rho\rangle$) for proving the security of $\Pi_{\mathsf{vpe}}\langle\rho\rangle$.

the value of $\widehat{x}_{i'}$ itself should be the same in all extractions.

This does not cover the possibility that if the root is corrupt, it could partition the honest parties into two sets and run independent executions of ρ with them (which is not allowed in the ideal world). It is to prevent this that we use a multi-signature in the commitment phase. Consider a corrupt party i' in a tree T_1 in the forest of corrupt parties, where T_1 is rooted at 1, the root of T (T_1 exists only if 1 itself was corrupt). In this case $\widehat{x}_{i'}$ extracted during the commitment phase is bound to α. We defined the input extraction (in Figure 6.2) using an arbitrary honest party P_k such that in T, k is adjacent to a leaf of T_1. But the unforgeability of the multi-signature ensures that all honest parties must agree on the same α. Hence, during the execution phase again, the extracted $\widehat{x}_{i'}$ is bound to the same α, no matter which honest party is the receiver. As before, due to the binding of COM and collision-resistance of Hash, this ensures that the value of $\widehat{x}_{i'}$ itself is the same in all extractions.

CHAPTER 7: CONCLUSION

BBT 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.[1] 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 several famous transformations (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 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 BBT 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 4.1). (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 [11] as a consequence of the earlier work of Ben-Or et al. and Chaum et al. [9, 10], via the above transformation.

2. We obtain the first "constant-rate" MPC protocol scheme with guaranteed output

[1] 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.

delivery against corruption of less than $n/2$ parties, provided the number of parties is constant (Corollary 4.2). 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 and C but dependent only on the number of parties. This result is obtained – following the lead of [14][2] – by applying our transformation to the scheme of [42] (combined with a secret-sharing scheme due to [43]) and the semi-honest secure scheme of [9].

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 4.6). By applying this transformation to the above protocol with full-security and an efficient protocol with security-with-abort from [40], we obtain a "scalable" MPC protocol with full-security and optimal corruption-threshold – i.e., tolerating corruption of less than $n/2$ parties (Corollary 4.3).[3] 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 [41], 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 [37] for the case of static zero-knowledge. (We note that the IKOS transformation for protocols in such hybrid models requires at least 3 parties.) The transformation from [37] 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 be applied towards efficient zero-knowledge proofs for *arithmetic* circuits and in both hybrids our transformation requires just a constant number of commitments overall (for a constant soundness error). This transformation may have relevance to the recent line of work on practical zero-knowledge proofs

[2] In [14], 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.

[3] 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).

initiated in [76]. In contrast to [37], we do not consider here the goal of adaptive zero-knowledge in the plain model.

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 4.4). This improves over a result of [36].[4]

   For this, we apply Bracha's transformation [18] 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 [36]), 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.

BBT 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 [35]. (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 3.1).

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 3.2).

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).

New Efficiency Measure: (MPC) Bottleneck Complexity. We introduce a new measure of per-party communication complexity for (distibuted) functions, called bottleneck

[4] In [36], 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.

complexity that measures the maximum communication required by any party within the protocol execution. While achieving $O(n)$ bottleneck complexity (where n is the number of parties) is straightforward, we show that achieving *sublinear* bottleneck complexity is *not* always possible, even when no security is required. We prove this by demonstrating the existence of n-party functions with k bits of input for each party, that have bottleneck complexity $\Theta(nk)$. Showing an explicit function with $\Omega(n)$ bottleneck complexity will require showing an explicit function with $\Omega(n^2)$ circuit size complexity. On the other hand, we observe that many useful classes of functions do have $o(n)$ bottleneck complexity.

MPC Transformation of Bottleneck Complexity. We show a general transformation from arbitrary efficient protocols to secure MPC protocols (in a model with public setup) that asymptotically (as a function of n) preserves the communication and computational requirements per party, and preserves the same communication graph. As part of our transformation, we introduce cryptographic primitives—Incremental FHE, Verifiable Protocol Execution—and give a construction of ZK-SNARKs with an ID-based simulation-extractability property. These may be of independent interest.

7.1 FUTURE WORK

The multiple dimensions of multiparty computation suggests a lot more potential topics to discuss. In addition to alternate communication measures such as the bottleneck complexity proposed in Chapter 5, alternate security guarantees such as the security with partially-identifiable-abort proposed in Chapter 4, and various thresholds of the corruption such as $1/2, 1/3$, an arbitrary constant fraction, etc., there are several more features to explore.

Black-box Transformations. Forgoing the binary corruption model for players (corrupt or honest), the line of rational cryptography models the parties as (mixed) rational players in games, and relies on designing mechanisms and equilibria for security. Black-box transformations across models involving various kinds of rational and malicious adversaries opens up interesting avenues to explore.

While the black-box transformation model – and some of the existing transformations – relate to protocols in hybrid models, our negative results consider black-box transformations of protocols in the plain model. This raises the question of how far the negative results hold in the hybrid models. In particular, given a semi-honest protocol in f-hybrid for a functionality drawn from a family, we may ask whether the protocol can be transformed into an actively secure protocol for the same functionality, in g-hybrid for interesting pairs of functionalities f and g. Further, one may extent the black-box transformation model

to a setting where the hybrid functionality f itself is drawn from a family and given as a black-box to the transformation.

Bottleneck Complexity. In Chapter 6, the adversary is static – i.e., it needs to fix a corruption set of parties beforehand. Furthermore, the results in Chapter 6 concern only protocols with a fixed communication pattern and leave out more general types of protocols involving dynamic communication patterns. Extending the results to those settings remains open.

Also, the results in Chapter 6 rely on SNARKs where the extractor incurs only an additive overhead, which is a strong assumption. This is crucial because when the overhead of extraction is multiplicative in the complexity of the proof, the prover's running time would grow exponentially – unless the protocol has a small number of communication rounds. We leave it open to obtain a better trade-off between the bottleneck complexity, the assumptions and the round complexity of the given protocols.

Finally, the cryptographic primitives that we developed, such as Incremental FHE, Verifiable Protocol Execution, and ZK-SNARKs with an ID-based simulation-extractability property, are potentially of independent interest. We leave it for future work to use them and build upon them.

REFERENCES

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

[2] O. Goldreich, S. Micali, and A. Wigderson, "How to play ANY mental game," in *Proc.* 19*th STOC*, ACM, Ed. ACM, 1987, see [16, Chap. 7] for more details. pp. 218–229.

[3] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, Nov. 1979.

[4] A. C. Yao, "Protocols for secure computation," in *Proc.* 23*rd FOCS*. IEEE, 1982, pp. 160–164.

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

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

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

[8] A. C. Yao, "How to generate and exchange secrets," in *Proc.* 27*th FOCS*. IEEE, 1986, pp. 162–167.

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

[10] D. Chaum, C. Crépeau, and I. Damgård, "Multiparty unconditionally secure protocols," in *Proc.* 20*th STOC*. ACM, 1988, pp. 11–19.

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

[12] J. Kilian, "Founding cryptography on oblivious transfer," in *STOC*. ACM, 1988, pp. 20–31.

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

[14] Y. Ishai, M. Prabhakaran, and A. Sahai, "Founding cryptography on oblivious transfer - efficiently," in *CRYPTO*. Springer, 2008, pp. 572–591.

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

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

[17] 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*, 2014, pp. 369–386.

[18] G. Bracha, "An o(log n) expected rounds randomized byzantine generals protocol," *J. ACM*, vol. 34, no. 4, pp. 910–920, 1987.

[19] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *STOC*, 1987.

[20] M. Ben-Or and R. Cleve, "Computing algebraic formulas using a constant number of registers," *SIAM Journal on Computing*, vol. 21, no. 1, pp. 54–58, 1992.

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

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

[23] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *EUROCRYPT*, 2003, pp. 416–432.

[24] S. Micali, K. Ohta, and L. Reyzin, "Accountable-subgroup multisignatures: extended abstract," in *ACM Conference on Computer and Communications Security*, 2001, pp. 245–254.

[25] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme," in *Public Key Cryptography*, 2003, pp. 31–46.

[26] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters, "Sequential aggregate signatures and multisignatures without random oracles," in *EUROCRYPT*, 2006, pp. 465–485.

[27] B. Waters, "Efficient identity-based encryption without random oracles." in *EUROCRYPT*, ser. Lecture Notes in Computer Science, R. Cramer, Ed., vol. 3494. Springer, 2005, pp. 114–127.

[28] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again," in *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, 2012, pp. 326–349.

[29] N. Bitansky, R. Canetti, O. Paneth, and A. Rosen, "On the existence of extractable one-way functions," in *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, 2014, pp. 505–514.

[30] E. Boyle and R. Pass, "Limits of extractability assumptions with distributional auxiliary input," in *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Part II*, 2015, pp. 236–261.

[31] P. Mukherjee and D. Wichs, "Two round multiparty computation via multi-key FHE," in *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, 2016, pp. 735–763.

[32] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Crypto'13*, 2013, pp. 75–92.

[33] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," in *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, 2005, pp. 84–93.

[34] D. Micciancio and C. Peikert, "Trapdoors for lattices: Simpler, tighter, faster, smaller," in *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, 2012, pp. 700–718.

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

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

[37] C. Hazay and M. Venkitasubramaniam, "On the power of secure two-party computation," Cryptology ePrint Archive, Report 2016/074, http://eprint.iacr.org/2016/074, 2016, to appear in *Proc. Crypto 2016*.

[38] Y. Ishai, M. Prabhakaran, and A. Sahai, "Secure arithmetic computation with no honest majority," in *TCC*, ser. Lecture Notes in Computer Science, O. Reingold, Ed., vol. 5444. Springer, 2009, pp. 294–314.

[39] 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*, 2005, pp. 79–99.

[40] 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.

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

[42] I. Damgård and Y. Ishai, "Scalable secure multiparty computation," in *Proc. 26th CRYPTO*. Springer, 2006, pp. 501–520.

[43] H. Chen and R. Cramer, "Algebraic geometric secret sharing schemes and secure multiparty computations over small fields," in *CRYPTO*. Springer, 2006, pp. 521–536.

[44] 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*. Springer, 2008, pp. 471–488.

[45] S. Cabello, C. Padró, and G. Sáez, "Secret sharing schemes with detection of cheaters for a general access structure," *Des. Codes Cryptography*, vol. 25, no. 2, pp. 175–188, 2002.

[46] S. P. Vadhan, "Pseudorandomness," *Foundations and Trends in Theoretical Computer Science*, vol. 7, no. 1–3, pp. 1–336, 2011.

[47] M. N. Garofalakis, J. Gehrke, and R. Rastogi, Eds., *Data Stream Management - Processing High-Speed Data Streams*, ser. Data-Centric Systems and Applications. Springer, 2016.

[48] A. McGregor, "Graph stream algorithms: a survey," *SIGMOD Record*, vol. 43, pp. 9–20, 2014.

[49] E. Kushilevitz and N. Nisan, *Communication complexity*. Cambridge University Press, 1997.

[50] M. Naor and K. Nissim, "Communication preserving protocols for secure function evaluation," in *Proceedings on 33rd Annual ACM Symposium on Theory of Computing (STOC)*, 2001, pp. 590–599.

[51] I. Damgård and Y. Ishai, "Scalable secure multiparty computation," in *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, 2006, pp. 501–520.

[52] I. Damgård, Y. Ishai, M. Krøigaard, J. B. Nielsen, and A. D. Smith, "Scalable multiparty computation with nearly optimal work and resilience," in *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, 2008, pp. 241–261.

[53] V. Dani, V. King, M. Movahedi, and J. Saia, "Quorums quicken queries: Efficient asynchronous secure multiparty computation," in *Distributed Computing and Networking - 15th International Conference, ICDCN 2014, Coimbatore, India, January 4-7, 2014. Proceedings*, 2014, pp. 242–256.

[54] E. Boyle, K. Chung, and R. Pass, "Large-scale secure computation: Multi-party computation for (parallel) RAM programs," in *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, 2015, pp. 742–762.

[55] M. Zamani, M. Movahedi, and J. Saia, "Millions of millionaires: Multiparty computation in large networks," *IACR Cryptology ePrint Archive*, vol. 2014, p. 149, 2014.

[56] N. Braud-Santoni, R. Guerraoui, and F. Huc, "Fast byzantine agreement," in *ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22-24, 2013*, 2013, pp. 57–64.

[57] E. Boyle, S. Goldwasser, and S. Tessaro, "Communication locality in secure multiparty computation: how to run sublinear algorithms in a distributed setting," in *Proceeding TCC'13 Proceedings of the 10th theory of cryptography conference on Theory of Cryptography*, 2013, pp. 356–376.

[58] I. Damgård, J. B. Nielsen, A. Polychroniadou, and M. Raskin, "On the communication required for unconditionally secure multiplication," in *Crypto'16*, 2016, pp. 459–488.

[59] I. Haitner, Y. Ishai, E. Kushilevitz, Y. Lindell, and E. Petrank, "Black-box constructions of protocols for secure computation," *SIAM J. Comput.*, vol. 40, no. 2, pp. 225–266, 2011.

[60] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs, "Multiparty computation with low communication, computation and interaction via threshold fhe," in *EUROCRYPT*, 2012, pp. 483–501.

[61] Y. Dodis, S. Halevi, R. D. Rothblum, and D. Wichs, "Spooky encryption and its applications," in *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III*, 2016, pp. 93–122.

[62] A. Sahai, "Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security," in *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, 1999, pp. 543–553.

[63] A. D. Santis, G. D. Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai, "Robust non-interactive zero knowledge," in *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, 2001, pp. 566–598.

[64] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," in *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, 2012, pp. 1219–1234.

[65] M. Clear and C. McGoldrick, "Multi-identity and multi-key leveled FHE from learning with errors," in *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, 2015, pp. 630–656.

[66] Z. Brakerski and R. Perlman, "Lattice-based fully dynamic multi-key FHE with short ciphertexts," in *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, 2016, pp. 190–213.

[67] C. Peikert and S. Shiehian, "Multi-key FHE from lwe, revisited," in *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, 2016, pp. 217–238.

[68] A. Chiesa and E. Tromer, "Proof-carrying data and hearsay arguments from signature cards," in *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceedings*, 2010, pp. 310–331.

[69] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "Recursive composition and bootstrapping for SNARKS and proof-carrying data," in *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, 2013, pp. 111–120.

[70] A. E. Kosba, Z. Zhao, A. Miller, Y. Qian, T. H. Chan, C. Papamanthou, R. Pass, A. Shelat, and E. Shi, "How to use snarks in universally composable protocols," *IACR Cryptology ePrint Archive*, vol. 2015, p. 1093, 2015. [Online]. Available: http://eprint.iacr.org/2015/1093

[71] D. Fiore and A. Nitulescu, "On the (in)security of snarks in the presence of oracles," in *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part I*, 2016, pp. 108–138.

[72] J. Groth and M. Maller, "Snarky signatures: Minimal signatures of knowledge from simulation-extractable snarks," in *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, 2017, pp. 581–612.

[73] C. Garman, M. Green, and I. Miers, "Accountable privacy for decentralized anonymous payments," in *Financial Cryptography and Data Security - 20th International Conference, FC 2016, Christ Church, Barbados, February 22-26, 2016, Revised Selected Papers*, 2016, pp. 81–98.

[74] I. Damgård, S. Faust, and C. Hazay, "Secure two-party computation with low communication," in *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*, 2012, pp. 54–74.

[75] J. Rompel, "One-way functions are necessary and sufficient for secure signatures," in *Proc. 22nd STOC.* ACM, 1990, pp. 387–394.

[76] 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*, 2013, pp. 955–966.