Cryptographic Algorithms for the Secure Delegation of Multiparty Computation

by

Adriana López-Alt

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Computer Science
New York University
May, 2014

Professor Yevgeniy Dodis

"It never ceased to amaze me, until suddenly one day I felt beautiful and holy for having had the courage to hold on to my sanity after all I'd seen and been through, body and soul, in too loud a solitude, and slowly I came to the realization that my work was hurtling me headlong into an infinite field of omnipotence."

– Bohumil Hrabal

To Kenji,
for loving a crypto nerd.

Para el phat man,
que me entrenó en mis primeras olimpiadas.

Para mi abuelito,
que predijo la secada de mis sesos.

Acknowledgements

I would like to thank my advisor, Yevgeniy Dodis, for all his support and encouragement, and for sharing his (I believe unmatched) enthusiasm for research. I especially thank him for teaching me the importance of good definitions and abstractions, and for teaching me to communicate my research, both in papers and in talks, in a way that comprehensibly conveys the importance of the results.

I would also like to thank all my internship hosts: my mentor Juan Garay at AT&T Research Labs (now at Yahoo Labs), for insisting I study lattices (admittedly against my will) and for giving me invaluable career advice; Ivan Damgård at Århus University, for hosting me when I was still new to crypto and research, and still had a lot to learn; Vinod Vaikuntanathan at the University of Toronto (now at MIT), for believing in me from the start, inexperienced as I was, and for introducing me to FHE; Yael Tauman Kalai at Microsoft Research in New England, for encouraging me to approach hard problems and take risks, and for showing me that an excellent research career and a wonderful personal life are not in any way paradoxical; Kristin Lauter and Michael Naehrig at Microsoft Research in Redmond, for showing me the great ways in which mathematicians and computer scientists can work together, and especially for showing me how much fun research can be; Úlfar Erlingsson and Tyler Close at Google, for showing me how to use my knowledge in theoretical cryptography to solve really cool problems in industry. I would also like to thank my mentor, Mary Fernandez, for helping me gain confidence in myself and in my research, and for supporting me through all aspects of Ph.D. and academic life.

I would like to thank my parents, Álvaro and Martha Lucía, for always making my education a priority, and for teaching me that anything can be accomplished with hard work; my sister Ana María, for teaching me the wonders of positive thinking; my brother Luis Eduardo, master of all negotiations, for making me laugh when I needed it the most; Alicia, Gloria, and Ernesto, for spoiling me rotten; Marcela and Dave, for welcoming me into their home. I especially thank my family for their constant encouragement and support, and for believing in me, unconditionally. Special thanks to Sebastián, for providing me with smiles and 'piruetas', which fueled me in the writing of this dissertation. I would also like to thank Valerio Pastro, for many great conversations, both research and non-research related; Emily Murray, for having an endless supply of tea, prosecco, and good advice; and Chichi Wang, for distracting me with fascinating stories, both true and fictional.

Most importantly, I would like to thank Kenji, for standing by me, unflinchingly, through all my insecurities, late nights, and endless travel, and for celebrating with me every published paper and every new thing learned.

This dissertation is based on two works, co-authored with Eran Tromer and Vinod Vaikuntanathan [LTV11, LTV12]. The first was published in EUROCRYPT 2012 [AJL+12], merged with the work of Asharov, Jain, and Wichs [AJW11]. The second was published in STOC 2012. I would like to thank all of my co-authors for their enthusiasm and hard work.

Abstract

In today's world, we store our data and perform expensive computations remotely on powerful servers (a.k.a. "the cloud") rather than on our local devices. In this dissertation we study the question of achieving cryptographic security in the setting where multiple (mutually distrusting) clients wish to delegate the computation of a joint function on their inputs to an untrusted cloud, while keeping these inputs private. We introduce two frameworks for modeling such protocols.

1. The first, called *cloud-assisted multiparty computation (cloud-assisted MPC)*, builds on the standard notion of MPC to incorporate the concept of delegation. In particular, since the cloud is expected to perform the computation of the function, our definition requires the communication complexity of the protocol, as well as the computation time of all clients to be (essentially) independent of the complexity of the function.

2. The second, called *on-the-fly MPC*, builds on the notion of cloud-assisted MPC and further requires that the clients be involved only when initially uploading their input to the cloud, and in a final phase when outputs are revealed. In particular, this allows the server to dynamically choose functions (and subsets of data on which to evaluate these functions) "on-the-fly", and evaluate them without requiring any interaction with the clients. The only interaction required takes place in the final phase after the computation has been completed, when the clients must retroactively approve both the chosen functions, and the subsets of data upon which these functions were evaluated.

We construct cloud-assisted and on-the-fly MPC protocols using fully homomorphic encryption (FHE). However, FHE requires inputs to be encrypted under the same key; we extend it to the multiparty setting in two ways:

1. We introduce the notion of *threshold FHE*: fully homomorphic encryption that allows the clients to jointly generate a common public key (whose corresponding secret key is shared among them), as well as decrypt a ciphertext under this public key without learning anything but the plaintext. Using threshold FHE, we show how to construct an efficient cloud-assisted MPC protocol. We construct threshold FHE using (a modification of) the Brakerski-Vaikuntanathan (ring-based) FHE scheme; however our ideas extend to many other lattice-based FHE schemes in the literature.

2. We introduce the notion of *multikey FHE*: fully homomorphic encryption that allows the cloud to perform homomorphic evaluation on ciphertexts encrypted under different and independent keys. We show a construction of on-the-fly MPC using multikey FHE, and construct a multikey FHE scheme based on NTRU encryption. We highlight that it was previously not known how to make NTRU fully homomorphic, even for a single key. Therefore, we view the construction of (multikey) FHE from NTRU encryption as a main contribution of independent interest.

Contents

Chapter 1
Introduction

We are fast approaching a new digital era in which we store our data and perform our expensive computations remotely, on powerful "cloud" servers, instead of locally, on our personal devices. This general model of cloud computing offers numerous advantages in costs and functionality. In particular, users are able to *delegate* expensive computations on their data, obtaining the desired output with very limited computation cost. The same applies when multiple clients, using the same cloud provider, wish to compute a function on their joint data.

When the client data is sensitive and the cloud server is untrusted, the problem of *privately* delegating computation on this data becomes more complex. In the single-client setting, fully homomorphic encryption [RAD78, Gen09b] gives a simple template for outsourcing computation. In this dissertation we study the analogous multi-client problem of *privately* delegating the *joint* computation of a function on the data of *multiple mutually distrusting* clients.

1.1 Overview of Results

We introduce two new notions of secure multiparty computation (MPC) [GMW87, BGW88, CCD88] that model the setting of delegation of multiparty computation to a powerful cloud server.

1.1.1 Cloud-Assisted MPC

The first new notion, which we call *cloud-assisted MPC*, imposes certain performance requirements on the protocol. In particular, it requires the communication complexity of the protocol, as well as the computation time of all clients, to be essentially independent of the complexity of the the desired computation. On the other hand, it allows the computation time of the server to be polynomial in the complexity of the function. Thus, the server is guaranteed to do all the "heavy lifting". We believe this captures the idea of *delegating* the *entire* computation to the server. We refer the reader to Section 3.1 for a formal definition.

Construction. Following Gentry's blueprint for constructing MPC from FHE, we are able to show an *efficient* construction of cloud-assisted MPC from *threshold FHE*: fully homomorphic encryption that allows parties to efficiently compute a joint public key pk from their individual public keys, and which allows efficient distributed decryption of a ciphertext encrypted under this joint public key pk.

We further show how to construct threshold FHE based on the Ring-LWE assumption of Lyubashevsky, Peikert, and Regev [LPR10]. For this, we use (a modification of) the Brakerski-Vaikuntanathan FHE scheme [BV11b]. We remark that our ideas are fairly general and can be extended to many other lattice-based FHE schemes in the literature [BV11a, BV11b, BGV12, Bra12, GSW13].

The following theorem gives an informal description of our results. See Chapter 4 for a formal theorem.

Theorem 1.1.1 (Informal)**.** *There exists a cloud-assisted MPC protocol in the CRS-model with the following properties:*

- *Runs in 4 rounds.*
- *Achieves security against malicious corruptions of an arbitrary subset of clients and possibly the server.*
- *The communication complexity of the protocol and the computation time of the clients is polylogarithmic in the size of the computation, and linear in the total size of the inputs and the size of the output.*
- *The computation time of the server is polynomial in the size of the circuit.*

1.1.2 On-the-Fly MPC

We introduce a second new notion of MPC called *on-the-fly MPC*, which makes the same performance requirements as cloud-assisted MPC but makes additional functional requirements. In particular, the server must be able to *dynamically* choose functions "on-the-fly" and evaluate these functions on *any subset* of the clients' inputs, without interacting with the clients. The clients need only approve these choices retroactively, once the computation has taken place.

An on-the-fly MPC protocol is divided into two phases, an offline and an online phase. In the offline phase, clients send their inputs to the server, who *stores* them indefinitely. In this phase, clients only communicate with the server and need not know that other clients exist. An online phase begins once a a function and subset of inputs have been chosen. In this phase, we require the computation of the clients and the communication complexity to be (essentially) independent of the complexity of the function being computed, as before. Additionally, we require these measures to only depend on the size of the *subset* of inputs used in the computation, and not on the overall number of clients, which can potentially be very large.

Construction. We show how to construct on-the-fly MPC using *multikey FHE*: fully homomorphic encryption that is able to perform homomorphic evaluation on ciphertexts encrypted under different, independent keys. This is a new and stronger notion of FHE; standard FHE does not immediately satisfy this property.

Our main contribution is showing that for *any number of keys* N, there exists a *multikey fully homomorphic* encryption scheme that is able to handle homomorphic evaluation of ciphertexts encrypted under N keys. Our construction is based on the NTRU encryption scheme of Hoffstein, Pipher, and Silverman [HPS98], with the modifications of Stehlé and Steinfeld [SS11b]. We view the construction of this (multikey) FHE as an important contribution of independent interest, and refer the reader to Chapter 5 for more details.

Theorem 1.1.2 (Informal)**.** *For all $N \in \mathbb{N}$, there exists a fully homomorphic encryption scheme that can perform homomorphic evaluation on ciphertexts encrypted under at most N different keys. The size of the keys and ciphertexts in the scheme grow polynomially with N.*

Using this construction of multikey FHE, we are able to build on-the-fly MPC. The following theorem gives an informal description of our results. See Chapter 6 for a formal theorem.

Theorem 1.1.3 (Informal)**.** *There exists an on-the-fly MPC protocol in the CRS-model with the following properties:*

- *Achieves security against malicious corruptions of an arbitrary subset of clients and possibly the server.*
- *The offline phase runs in one (asynchronous) round of unidirectional communication from the parties to the server. The online phase runs in 5 rounds.*
- *The communication complexity of the online phase and the computation time of the computing parties therein is polylogarithmic in the size of the computation and the total size of the inputs, and linear in the size of their own input and the size of the output.*
- *The computation time of the server is polynomial in the size of the circuit.*

1.2 Cloud-Assisted MPC

As described in Section 1.1.1, we formalize the efficiency requirements imposed by the notion of delegating multiparty computation, by extending the notion of MPC to what we call *cloud-assisted MPC*. In cloud-assisted MPC, the communication of the protocol, as well as the computation time of all clients is required to be essentially[1] independent of the circuit-complexity of the function being computed. On the other hand, the computation time of the server is allowed to be polynomial in the size of the circuit. We believe this captures the idea of *delegating* the *entire* computation to the server.

Single-Client Delegation of Computation. Consider the simplified setting in which a *single* client wants to privately delegate to a server the computation of a function on her input. Fully homomorphic encryption (FHE) [RAD78, Gen09b] provides a mechanism for the client to do precisely this. In short, the client samples an encryption key, encrypts her input, and sends the result to the server along with a public evaluation key, if one is required. The server then homomorphically evaluates the desired function on the ciphertext and returns the result to the client, who can decrypt it to obtain the unencrypted output of the computation. As long as the server is semi-honest, this basic construction provides privacy and correctness. When the server is malicious, techniques from verifiable computation [GGP10, CKV10, AIK10] and succinct argument systems [Kil92, Kil95, Mic94, GKR08, GLR11, BCCT12, BCCT13, Gro10, Lip12, GGPR13, PHGR13, Lip13] can be employed to guarantee that the server performs the computation as prescribed. Moreover, since the groundbreaking result of Gentry [Gen09b, Gen09a], many improvements and new constructions of FHE have appeared in the literature [vDGHV10, BV11b, BV11a, BGV12, Bra12, BLLN13, GSW13, BV14], making this a feasible solution. The question we pose is the following:

Question 1: Is it possible to achieve cloud-assisted MPC by extending this simple template to the setting where *multiple* parties wish to delegate the computation of a *joint* function on their inputs?

[1] We allow these measures to depend *polylogarithmically* on the circuit-size of the function. This is necessary in order to guarantee security against a malicious server.

MPC from FHE: Gentry's Blueprint. Gentry [Gen09a] showed that the answer to this question is *yes*. In short, he described how to construct an MPC protocol for any efficiently computable function, using FHE. Gentry's construction runs in 4 steps:

Step 1: Parties run a generic MPC protocol to obtain a public key pk and an evaluation key ek for an FHE scheme, as well as secret shares $\mathsf{sk}_1, \ldots, \mathsf{sk}_N$ of a corresponding secret key sk.

Step 2: Each party encrypts their input m_i under public key pk and broadcasts the resulting ciphertext c_i.

Step 3: Each party individually and locally performs the homomorphic evaluation $c \stackrel{\mathsf{def}}{=} \mathsf{Eval}(\mathsf{ek}, C, c_1, \ldots, c_N)$

Step 4: Parties run a generic MPC protocol to decrypt c using their shares sk_i, and obtain the output of the computation.

A simple observation is that the homomorphic computation in Step 3 can be performed by a single party, the server or "cloud". Gentry's construction can then viewed as one in which the multiple clients *emulate* the client in the single-client protocol described above.

Main Questions. Gentry's construction shows how to achieve cloud-assisted MPC using FHE and generic MPC techniques, giving a positive answer to Question 1. While the construction guarantees that the communication complexity of the protocol and the computation time of all parties except the server, is essentially independent of the complexity of the function being computed, the addition of generic MPC has two downsides: first, these measures can still be quite high, and second, the round complexity of the overall protocol can be very large. We therefore ask the following two questions:

QUESTION 2: Is it possible to achieve cloud-assisted MPC without using generic MPC techniques?

QUESTION 3: Can we further reduce the round complexity of the protocol? In particular, can we achieve *constant* round complexity?

As one of our main results, we give a *positive* answer to both of these questions. The following theorem gives an informal description of the result, and in Section 1.2.1 we give an overview of the techniques used to achieve it. In Chapter 4, we show a detailed construction and a give a formal theorem.

Theorem 1.2.1 (Informal)**.** *There exists a cloud-assisted MPC protocol in the CRS-model with the following properties:*

- *Runs in 4 rounds.*
- *Achieves security against malicious corruptions of an arbitrary subset of clients and possibly the server.*
- *The communication complexity of the protocol and the computation time of the clients is polylogarithmic in the size of the computation, and linear in the total size of the inputs and the size of the output.*
- *The computation time of the server is polynomial in the size of the circuit.*

1.2.1 Our Techniques

We construct the protocol described in Theorem 1.2.1 by leveraging specific *key-homomorphic* properties of (Ring-)LWE-based FHE schemes [BV11a, BV11b, BGV12, Bra12, GSW13].[2] For example, in Ring-LWE based schemes we work over a polynomial ring. A system parameter is created by sampling a uniformly random ring element a. The secret key is a "small" ring element s, and the public key is computed by sampling a "small" noise element e and computing $p = [as + e]_q$, where $[\ \cdot\ ]_q$ denotes coefficient-wise reduction modulo q into the set $\left\{-\left\lfloor \frac{q}{2} \right\rfloor, \ldots, \left\lfloor \frac{q}{2} \right\rfloor\right\}$.

A simple observation yields the following fact: Summing two public keys under the *same* system parameter yields a *valid* public key for the *sum* of the underlying secret keys: if $p_1 = [as_1 + x_1]_q$ and $p_2 = [as_2 + x_2]_q$ are two public keys, then:

$$[p_1 + p_2]_q = [a(s_1 + s_2) + (x_1 + x_2)]_q$$

Therefore $[p_1 + p_2]_q$ is a valid public key for secret key $s_1 + s_2$.

Joint-Key Computation. Our main observation is that using a key-homomorphic FHE scheme in Gentry's blueprint allows us to bypass the expensive MPC protocol required to compute a joint public key (Step 1). Instead, each party computes its own key pair (p_i, s_i), independent of all others, and broadcasts its public key p_i. In Step 2, the parties can add all public keys to obtain the joint public key $p = [p_1 + \ldots + p_N]_q$, whose corresponding joint secret key $s = s_1 + \ldots + s_N$, is additively secret-shared among them.

When the FHE scheme does not require the use of an evaluation key to perform homomorphic operations, as in the case of the recent scheme of Gentry, Sahai, and Waters [GSW13], computing a joint public key suffices. However, all other FHE schemes being considered [BV11a, BV11b, BGV12, Bra12] do use an evaluation key. For these we also require a method for computing a joint evaluation key. This turns out to be a trickier problem, for which we currently do not know a non-interactive solution. However, we show a 2-round protocol to achieve it. We remark that in both cases, the resulting MPC protocol has the same number of rounds, namely 4.

Distributed Decryption. We take advantage of the same linear structure of the scheme to do very efficient (1-round) distributed decryption in the style of Bendlin and Damgård [BD10]. This allows us to further bypass the expensive MPC protocol required to perform joint decryption (Step 3). In essence, to jointly decrypt a ciphertext c, each party broadcasts a *decryption share* μ_i. The ciphertext can be combined with all decryption shares to obtain the underlying plaintext.

(Informal) Protocol: Our protocol can then be informally summarized as follows:

Round 1: Each party samples independent key tuples (p_i, s_i) and broadcasts the public key p_i. If necessary, it also broadcasts the first round in the 2-round protocol for computing the joint evaluation key.

Round 2: Each party computes the joint public key p, encrypts their input under p and broadcasts the ciphertext. If necessary, it also broadcasts the second round in the 2-round protocol for computing the joint evaluation key.

[2] Key-homomorphic properties are by no means unique to lattice-based schemes; many number-theoretic constructions exhibit the same structure, e.g. the ElGamal encryption scheme [Gam84]. However, this property is especially useful when displayed in fully homomorphic schemes, as we will demonstrate in this work.

Round 3: The server computes the joint evaluation key, and performs the homomorphic evaluation. When finished, it broadcasts the evaluated ciphertext c to all parties.

Round 4: Each party computes and broadcasts its decryption share μ_i with its individual secret key s_i.

Local Computation: Each party combines the ciphertext c with all the decryption shares to obtain the output of the computation.

Security. We show that the above protocol is secure against *semi-malicious* adversaries [AJW11, AJL+12], who follow the protocol specifications (like semi-honest adversaries) but choose their random coins from an arbitrary distribution (like malicious adversaries). We then show how to convert the protocol into one that is secure against malicious corruptions by using the AJW compiler [AJW11, AJL+12] from semi-malicious to malicious security, which is based on the GMW compiler [GMW87] from semi-honest to malicious security. This requires the parties to prove in zero-knowledge that their message at every round is consistent with the protocol transcript so far. To obtain security against a malicious server, we must also rely on techniques from verifiable computation [GGP10, CKV10, AIK10] or succinct argument systems [Kil92, Kil95, Mic94, GKR08, GLR11, BCCT12, BCCT13, Gro10, Lip12, GGPR13, PHGR13, Lip13] to ensure that the server performs the homomorphic computation correctly.

This approach differs from the one presented in the original version of our work [LTV11] and instead follows the outline of Asharov et al. [AJW11, AJL+12]. The main difference is in the realization that the protocol is *already* secure against corrupt parties that choose their random coins adversarially. In our previous work [LTV11], we used *unnecessary* coin-flipping techniques in order to guarantee this.

Other Instantiations. We remark that the ideas behind our protocol are fairly general and can be instantiated with a number of different existing FHE schemes [BV11a, BV11b, BGV12, Bra12, GSW13]. For clarity of presentation, we choose to present it based on the Ring-LWE-based FHE scheme of Brakerski and Vaikuntanathan [BV11b], using relinearization/key-switching [BV11a] and the noise-management technique of Brakerski [Bra12]. The scheme resulting from the combination of these techniques has already been described by Fan and Vercauteren [FV12].

1.3 On-The-Fly MPC

Having shown how to construct cloud-assisted MPC allowing several mutually distrusting parties to delegate a computation on their joint inputs to an untrusted but powerful server, we consider a more stringent model that has the same efficiency requirements, but allows more flexibility in the computation of functions.

In *on-the-fly* MPC, clients originally upload their (possibly very large) inputs to the cloud in an offline stage, before any computation takes place, before even a decision to compute on this data is made. This captures the notion of *delegation of storage*, in which users store their data remotely on "cloud" servers rather than locally on personal computers. In this offline stage, clients are not aware of other parties uploading their data to the server; in particular, we require that no interaction between clients take place at this stage. We are interested in the setting in which the *universe* of clients that upload their data to the server is large, in the order of millions or more.

After this initial offline phase, the server can *dynamically* select functions "on-the-fly" and evaluate them on the clients' inputs without requiring any interaction from the clients. Indeed, we require that the clients be able to be offline or "asleep" during the entire computation, and only "wake-up" to retroactively approve the function that was computed on their input. Moreover, the selected function can be computed on the inputs of any *subset* of the universe of parties, and the complexity of the resulting online protocol must only depend on the size of the subset, not on the size of the universe, which as we said can be quite large.

Having defined the model (see Section 3.2 for a more formal definition), the question becomes:

QUESTION 4: Can we construct an MPC protocol that satisfies the strict requirements of on-the-fly MPC?

As one of our a main contributions, we answer this question *positively*: we construct an on-the-fly MPC protocol for any efficiently-computable function. The following theorem gives an informal description of the result, and in the remainder of this section we give an overview of the techniques used to achieve it. In Chapter 6, we show a detailed construction and a give a formal theorem.

Theorem 1.3.1 (Informal)**.** *There exists an on-the-fly MPC protocol in the CRS-model with the following properties:*

- *Achieves security against malicious corruptions of an arbitrary subset of clients and possibly the server.*
- *The offline phase runs in one (asynchronous) round of unidirectional communication from the parties to the server. The online phase runs in 5 rounds.*
- *The communication complexity of the online phase and the computation time of the computing parties therein is polylogarithmic in the size of the computation and the total size of the inputs, and linear in the size of their own input and the size of the output.*
- *The computation time of the server is polynomial in the size of the circuit.*

Having constructed on-the-fly MPC with a constant number of rounds in the online stage, it is natural to ask if the optimal round complexity is feasible:

QUESTION 5: Is it possible to obtain the optimal solution of a protocol with a completely non-interactive online phase (namely, a protocol where the users do not ever have to talk to each other, even in the decryption phase)?

We know from the work of Halevi, Lindell, and Pinkas [HLP11] that in the non-interactive setting, the server can always evaluate the circuit multiple times, keeping some parties inputs but plugging in fake inputs of its choosing for the other parties. However, even if we accept this as the ideal functionality, we show that a non-interactive online phase *cannot* be achieved by drawing on the impossibility of general program obfuscation as a virtual black-box with single-bit output [BGI$^+$01]. Thus, our notion is qualitatively "the best possible" in terms of interaction. Our techniques in showing this negative result are inspired by those of van Dijk and Juels [vDJ10]. We refer the reader to Section 6.3 for more details.

Theorem 1.3.2. *There* does not *exist an on-the-fly MPC protocol that displays a completely non-interactive online phase.*

1.3.1 Overview of our Construction.

Our construction of "on-the-fly" MPC is loosely based on Gentry's blueprint for constructing MPC from FHE [Gen09a], described in Section 1.2. In the first step, during the offline phase, the parties must send their inputs to the server. To guarantee privacy of their inputs, the natural approach is for the parties to encrypt them. The question becomes, encrypt them *under what key*? Recall that in this phase, no interaction can take place between clients, so Gentry's approach and our optimization using threshold FHE does not offer a solution. The only apparent solution is to have each party sample its own independent encryption key and encrypt their input under this key. However, this poses a problem in the online phase when the server must homomorphically evaluate a function on ciphertexts encrypted *under different and independent keys*, without interacting with the clients. Standard FHE does not guarantee successful evaluation in this setting. We thus ask the following question.

Question 6: Does there exist an FHE scheme that can homomorphically evaluate functions on ciphertexts encrypted under different, independent keys?

The answer to this question is *yes*. In particular, we show that *any* FHE scheme is inherently a *multikey FHE* for a constant number of keys (in the security parameter), i.e. it can homomorphically evaluate functions on ciphertexts encrypted under at most a constant number of keys.[3] Furthermore, we show that the Ring-LWE based FHE scheme of Brakerski and Vaikuntanathan [BV11b] is multikey homomorphic for a logarithmic number of keys, but only for circuits of logarithmic depth. This arises from the fact that when multiple keys are introduced, it is no longer clear how to use relinearization or squashing to go beyond somewhat homomorphism. We refer the reader to Section 5.2 for more details.

While these results are nice, to construct on-the-fly MPC we need more: we need a multikey FHE that can evaluate ciphertexts encrypted under *any* number of keys. This is because in the MPC setting, the number of parties N is fixed and the security parameter of the scheme can be at most polynomial in N. Thus, we ask:

Question 7: Does there exist a multikey FHE that can perform homomorphic evaluations on ciphertexts encrypted under *any number of keys*?

One of the most important contributions of this dissertation is giving a *positive* (but *partial*) answer to this question. We construct a multikey FHE for any number of keys based on the (modified) NTRU encryption scheme [HPS98, SS11b]. However, our construction requires a-priori knowledge of the number of keys; removing this dependence remains an interesting and seemingly challenging open problem.

The following theorem gives an informal description of the result, and in Section 1.3.2 we give an overview of the techniques used to achieve it. In Section 5.3, we give a detailed construction and show a formal theorem.

Theorem 1.3.3 (Informal)**.** *For all $N \in \mathbb{N}$, there exists a fully homomorphic encryption scheme that can perform homomorphic evaluation on ciphertexts encrypted under at most N different keys. The size of the keys and ciphertexts in the scheme grow polynomially with N.*

[3] This construction was originally suggested to us by an anonymous STOC 2012 reviewer; we include it here for completeness.

This restriction translates directly to our on-the-fly protocol: an a-priori bound on the number of *computing parties* (but not on the size of the universe) must be known a-priori.

1.3.2 (Multikey) Fully Homomorphic Encryption from NTRU

The starting point for our main construction of multikey FHE is the NTRU encryption scheme of Hoffstein, Pipher, and Silverman [HPS98], with the modifications of Stehlé and Steinfeld [SS11b]. NTRU encryption is one of the earliest lattice-based cryptosystems, together with the Ajtai-Dwork cryptosystem [AD97] and the Goldreich-Goldwasser-Halevi cryptosystem [GGH97]. One of our most important contributions is to show that NTRU can be made fully homomorphic (for a single key)[4] and moreover, that the resulting scheme can handle homomorphic evaluations on ciphertexts encrypted under any number of *different and independent* keys.

We find this contribution particularly interesting because NTRU was originally designed to be an *efficient* public-key encryption scheme, meant to replace RSA in applications where computational efficiency is at a premium (e.g. in applications that run on smart cards and embedded systems). Although the transformation to fully homomorphic encryption degrades the efficiency of the scheme, we believe it to be a *leading candidate* for a *practical* FHE scheme. Therefore, we view this as an important contribution of independent interest.

In this section we give an overview of our construction, and refer the reader to Section 5.3 for more details.

NTRU Encryption. We describe the modified NTRU scheme of Stehlé and Steinfeld [SS11b], which is based on the original NTRU cryptosystem [HPS98]. The scheme is parametrized by the ring $R \stackrel{\mathsf{def}}{=} \mathbb{Z}[x]/\langle x^n + 1\rangle$, where n is a power of two, an odd prime number q, and a B-bounded distribution χ over R, for $B \ll q$. By "B-bounded", we mean that the magnitude of the coefficients of a polynomial sampled from χ is guaranteed to be less than B. We define $R_q \stackrel{\mathsf{def}}{=} R/qR$, and once again use $[\ \cdot\ ]_q$ to denote coefficient-wise reduction modulo q into the set $\left\{-\left\lfloor \frac{q}{2} \right\rfloor, \ldots, \left\lfloor \frac{q}{2} \right\rfloor\right\}$.

- $\mathsf{Keygen}(1^\kappa)$: Key generation samples "small" polynomials $f', g \leftarrow \chi$ ands sets $f \stackrel{\mathsf{def}}{=} 2f' + 1$ so that $f \pmod 2 = 1$. If f is not invertible in R_q, it resamples f'. Otherwise, it computes the inverse f^{-1} of f in R_q and sets

$$\mathsf{sk} = f \qquad \text{and} \qquad \mathsf{pk} = \left[2gf^{-1}\right]_q$$

- $\mathsf{Enc}(\mathsf{pk}, m)$: To encrypt a bit $m \in \{0, 1\}$, the encryption algorithm samples "small" polynomials $s, e \leftarrow \chi$, and outputs the ciphertext

$$c = [hs + 2e + m]_q$$

- $\mathsf{Dec}(\mathsf{sk}, c)$: To decrypt a ciphertext c, the decryption algorithm computes $\mu = [fc]_q$ and returns $\mu \pmod 2$.

[4] The observation that NTRU can be made *single-key* fully homomorphic was made concurrently by Gentry et al.[GHL+11].

Correctness follows from a few simple observations. First note that $[fc]_q = [2gs + 2fe + fm]_q$. Furthemore, since the elements g, s, f, e were all sampled from a B-bounded distribution and $B \ll q$, the magnitude of the coefficients in $2gs + 2fe + fm$ is smaller than $q/2$, so there is no reduction modulo q: in other words, $[2gs + 2fe + fm]_q = 2gs + 2fe + fm$. Therefore, $\mu = 2gs + 2fe + fm$. Taking modulo 2 yields the message m since by construction, $f \equiv 1 \pmod 2$.

Multikey Homomorphism. We now briefly describe the (multikey) homomorphic properties of the scheme and the challenges encountered when converting it into a fully homomorphic encryption scheme.

Let $c_1 = [h_1 s_1 + e_1 + m_1]_q$ and $c_2 = [h_2 s_2 + e_2 + m_2]_q$ be ciphertexts under two different keys $h_1 = \left[2g_1 f_1^{-1}\right]_q$ and $h_2 = \left[2g_2 f_2^{-1}\right]_q$, respectively. We claim that $c_{\mathsf{add}} \stackrel{\mathsf{def}}{=} [c_1 + c_2]_q$ and $c_{\mathsf{mult}} \stackrel{\mathsf{def}}{=} [c_1 c_2]_q$ decrypt to $m_1 + m_2$ and $m_1 m_2$ respectively, under the *joint* secret key $f_1 f_2$. Indeed, notice that:

$$\begin{aligned} f_1 f_2 (c_1 + c_2) &= 2\left(f_1 f_2 e_1 + f_1 f_2 e_2 + f_2 g_1 s_1 + f_1 g_2 s_2\right) + f_1 f_2 (m1 + m_2) \\ &= 2e_{\mathsf{add}} + f_1 f_2 (m1 + m_2) \end{aligned}$$

for a slightly larger noise element e_{add}. Similarly,

$$\begin{aligned} f_1 f_2 (c_1 c_2) &= 2(2g_1 g_2 s_1 s_2 + g_1 s_1 f_2 (2e_2 + m_2) + g_2 s_2 f_1 (2e_1 + m_1) + \\ &\qquad f_1 f_2 (e_1 m_2 + e_2 m_1 + 2e_1 e_2)) + f_1 f_2 (m_1 m_2) \\ &= 2e_{\mathsf{mult}} + f_1 f_2 (m1 m_2) \end{aligned}$$

for slightly larger noise element e_{mult}. This shows that the ciphertexts $c_{\mathsf{add}} \stackrel{\mathsf{def}}{=} [c_1 + c_2]_q$ and $c_{\mathsf{mult}} \stackrel{\mathsf{def}}{=} [c_1 c_2]_q$ can be correctly decrypted to the sum and the product of the underlying messages, respectively, as long as the error does not grow too large.

Extending this to circuits, we notice that the secret key required to decrypt a ciphertext c that is the output of a homomorphic evaluation on ciphertexts encrypted under N different keys, is $\prod_{i=1}^{N} f_i^{d_i}$, where d_i is the degree of the ith variable in the polynomial function computed by the circuit. Thus, decrypting a ciphertext that was the product of a homomorphic evaluation requires knowing the circuit! This is unacceptable even for somewhat homomorphic encryption.

We employ the relinearization technique of Brakerski and Vaikuntanathan [BV11a], to essentially reduce the degree from d_i to 1, so that the key needed to decrypt the evaluated ciphertext is now $\prod_{i=1}^{N} f_i$. This guarantees that decryption is dependent on the number of keys N but independent of the circuit computed. After using relinearization, we can show that the resulting scheme is multikey somewhat homomorphic for $\approx n^{\delta}$ keys and circuits of depth $\approx \log\log q - \delta \log n$ for any $\delta \in (0, 1)$.

From (Multikey) Somewhat to Fully Homomorphic Encryption. Once we obtain a (multikey) somewhat homomorphic encryption scheme, we can apply known techniques to convert it into a (multikey) fully homomorphic scheme. In particular, we follow the original template of our work [LTV12] and use modulus reduction [BV11a, BGV12] to increase the circuit depth that the

scheme can handle in homomorphic evaluation. This yields a leveled homomorphic scheme for N keys that can evaluate circuits of depth D as long as $ND \approx \log q$. For any number of keys N and any depth D, we can set q to be large enough to guarantee the successful homomorphic evaluation of depth-D circuits on ciphertexts encrypted under N different keys.

Theorem 1.3.4 (Informal)**.** *For all $N \in \mathbb{N}$ and $D \in \mathbb{N}$, there exists a leveled homomorphic encryption scheme that can homomorphically evaluate depth-D circuits on ciphertext encrypted under at most N different keys. The size of the keys and ciphertexts in the scheme grow polynomially with N and D.*

Finally, using an analog of Gentry's bootstrapping theorem [Gen09b, Gen09a] for the multikey setting, we can convert the leveled homomorphic scheme into a fully homomorphic scheme, in which the algorithms are independent of the circuit depth D (albeit with an additional circular security assumption). On the other hand, we are unable to remove the dependence on the number of keys N, and therefore obtain a scheme that is fully homomorphic with respect to the depth of circuits it can evaluate, but "leveled" with respect to the number of different keys it can handle.

We remark that using the recent noise-management technique of Brakerski [Bra12], it is possible to obtain a simpler leveled homomorphic scheme, based on a weaker security assumption. This was already noted in the follow-up work of Bos et al. [BLLN13]. In another recent work, Gentry, Sahai, and Waters [GSW13] show how to remove the required evaluation key, yielding an even simpler scheme.

Security. Stehlé and Steinfeld [SS11b] showed that the security of the modified NTRU encryption scheme can be based on the Ring-LWE assumption of Lyubashevsky et al., which can be reduced to worst-case hard problems in ideal lattices [LPR10]. To prove the security of NTRU, Stehlé and Steinfeld first show that the public key $h = \left[2gf^{-1}\right]_q$ is *statistically* close to uniform over the ring R if f' and g are sampled from a discrete Gaussian with standard deviation $\mathsf{poly}(n)\sqrt{q}$ (which can be shown to be a $\mathsf{poly}(n)\sqrt{q}$-bounded distribution). Unfortunately, if we sample f' and g from this distribution the error in a *single* homomorphic operation would grow large enough to cause decryption failures. We must therefore make the *assumption* that the public key $h = \left[2gf^{-1}\right]_q$ is *computationally indistinguishable*[5] from uniform over R when f' and g are sampled from a discrete Gaussian that is B-bounded for $B \ll q$.

Ultimately, we arrive at the following theorem.

Theorem 1.3.5 (Informal)**.** *For all $N \in \mathbb{N}$, there exists a fully homomorphic encryption scheme that can perform homomorphic evaluation on ciphertext encrypted under at most N different keys. The size of the keys and ciphertexts in the scheme grow polynomially with N. The security of the scheme is based on the Ring-LWE assumption, the assumption that the public key is pseudorandom, and the assumption that the scheme is weakly circular secure.*

In a follow-up work, Bos et al. [BLLN13] show how to apply Brakerski's techniques [Bra12] to maintain the fully homomorphic properties of the scheme while sampling the elements f' and g from a discrete Gaussian with standard deviation $\mathsf{poly}(n)\sqrt{q}$, as in the work of Stehlé and Steinfeld [SS11b]. This yields an NTRU-based FHE scheme that is secure under the RLWE assumption alone.

[5] It is not difficult to see that with our setting of parameters, the distribution of the public key is not statistically close to uniform. We must therefore rely on *computational* indistinguishability.

However, as far as we know, this scheme is multikey for only a constant number of parties, which is an inherent property of any FHE scheme (see Section 5.2.1).

1.3.3 On-The-Fly MPC from Multikey FHE

Once we have constructed multikey FHE for any number of keys, we can construct on-the-fly MPC quite easily by loosely following Gentry's blueprint. The following gives an informal outline of our protocol.

Offline Phase: The clients sample independent key pairs $(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)$, encrypt their input under their corresponding public key: $c_i \leftarrow \mathsf{Enc}(\mathsf{pk}_i, x_i)$, and send this ciphertext to the server along with the public and evaluation keys $(\mathsf{pk}_i, \mathsf{ek}_i)$.

Online Phase: Once a function has been chosen, together with a corresponding subset of computing parties V:

Step 1: The server performs the *multikey homomorphic evaluation* of the desired circuit on the corresponding ciphertexts, and broadcasts the evaluated ciphertext to all computing parties (i.e. all parties in V).

Step 2: The computing parties (i.e. parties in V) run a generic MPC protocol to decrypt the evaluated ciphertext using their individual secret keys sk_i.

Observe that the computation of the decryption function in Step 2 of the online phase can itself be delegated to the server. In particular, if we instantiate the decryption protocol using our cloud-assisted MPC protocol (described in Section 1.2 and Chapter 4), we obtain a round-efficient solution: the overall protocol has an online phase of only 5 rounds.

1.3.4 Protocol Security

As in our cloud-assisted protocol, we show that the above protocol is secure against *semi-malicious* adversaries [AJW11, AJL+12], who follow the protocol specifications (like semi-honest adversaries) but choose their random coins from an arbitrary distribution (like malicious adversaries). But instead of using the AJW compiler [AJW11, AJL+12] to achieve security against malicious adversaries, we modify the protocol directly and prove its security. We make three modifications, described below.

Modifying the Decryption Protocol. The first modification we make is to change the decryption protocol in Step 2 of the online phase to first check that the secret key being used is a *valid* secret key for the corresponding public and evaluation keys. This ensures that if decryption is successful, then in particular, a corrupted party knows a valid secret key $\widetilde{\mathsf{sk}}_i$. This secret key *binds* the corrupted party to the input $\widetilde{x}_i = \mathsf{Dec}\left(\widetilde{\mathsf{sk}}_i, \widetilde{c}_i\right)$, which by semantic security of the FHE, must be independent of the honest inputs.

Once again, we note that the computation of this function can be delegated to the server using our cloud-assisted protocol (described in Section 1.2 and Chapter 4), yielding a 5-round online phase.

Adding Zero-Knowledge Proofs. We further require that in the offline phase, each party create a non-interactive zero-knowledge proof π_i^{ENC} showing that the ciphertext c_i is well-formed (i.e. that there exists plaintext x_i and randomness s_i such that $c_i = \mathsf{Enc}(\mathsf{pk}_i, x_i\ ;\ s_i)$. This guarantees that for a corrupted party, $\mathsf{Dec}\left(\widetilde{\mathsf{sk}}_i, \widetilde{c}_i\right) \neq \perp$ and thus the party really "knows" an input $\widetilde{x}_i$. Furthermore, it guarantees that the ciphertexts c_i are *fresh* encryptions, which is important in our setting of fully homomorphic encryption where we must ensure that the error stays low in a homomorphic evaluation.

While constructions of NIZK arguments are known for all of NP [GOS06, GOS12], using these constructions requires expensive NP reductions. To avoid this, in Section 6.2.6 we show how to construct an efficient NIZK argument system, secure in the random oracle model, for proving the well-formedness of a ciphertext in the NTRU-based multikey FHE scheme (the scheme we use to instantiate the generic multikey FHE scheme in our on-the-fly MPC construction).

Adding Verification of Computation. Finally, we must also rely on a succinct argument system [Kil92, Kil95, Mic94, GKR08, GLR11, BCCT12, BCCT13] to ensure that the server performs the homomorphic computation correctly. Due to the dynamic nature of our on-the-fly model, we are unable to use verifiable computation protocols in the pre-processing model [GGP10, CKV10, AIK10] or succinct arguments with a reference string that depends on the circuit being computed [Gro10, Lip12, GGPR13, PHGR13, Lip13]. These would require the clients to perform some pre-computation dependent on the circuit to be computed *before* knowing the circuit, or to interact with the server after a function has been selected and compute in time proportional to the circuit-size of the function. Indeed, the beauty of our on-the-fly MPC model is that the server can choose *any* function *dynamically*, "on-the-fly", and homomorphically compute this function *without* interacting with the clients, who additionally, compute in time only polylogarithmically in the size of any function being computed.

We show how to guarantee verification of computation in two different cases.

Verification for Small Inputs: When the total size of the inputs (and therefore the ciphertexts) is small enough to be broadcasted to all parties, it suffices for the server to use any of the succinct arguments of [Kil92, Kil95, Mic94, GKR08, GLR11, BCCT12, BCCT13] to prove that it carried out the computation correctly as specified. Along with this argument, the server broadcasts the ciphertexts c_i and public and evaluations keys $(\mathsf{pk}_i, \mathsf{sk}_i)$ for all parties in V. With this information, the computing parties can verify the argument before engaging in the decryption protocol.

Verification for Large Inputs: In the case when the total size of the inputs (and therefore the ciphertexts) is too large to be broadcasted to all parties, then we additionally require the parties to sample a hash key hk_i for a collision-resistant hash function, and compute a digest d_i of the ciphertext c_i. Each party then sends the tuple $(\mathsf{pk}_i, \mathsf{ek}_i, c_i, \pi_i^{\text{ENC}}, \mathsf{hk}_i, d_i)$ to the server in the offline phase. It is then sufficient for the server to broadcast the tuples $(\mathsf{pk}_i, \mathsf{ek}_i, \mathsf{hk}_i, d_i)$ and a succinct argument for the NP language:

> "there exist $\widetilde{c}_1, \widetilde{\pi}_1^{\text{ENC}}, \ldots, \widetilde{c}_N, \widetilde{\pi}_N^{\text{ENC}}$ such that $d_i = H_{\mathsf{hk}_i}(\widetilde{c}_i)$ and
> $c = \mathsf{Eval}(C, (\widetilde{c}_1, \mathsf{pk}_1, \mathsf{ek}_1), \ldots, (\widetilde{c}_N, \mathsf{pk}_N, \mathsf{ek}_N))$ and $\widetilde{\pi}_i^{\text{ENC}}$ is a valid proof".

If the succinct argument is additionally a *proof of knowledge*, as in the case of CS proofs [Mic94] under Valiant's analysis [Val08], and the SNARKs of Bitansky et al. [BCCT12, BCCT13],

then we are guaranteed that the server actually "knows" such $\widetilde{c}_1, \widetilde{\pi}_1^{\mathrm{ENC}}, \ldots, \widetilde{c}_N, \widetilde{\pi}_N^{\mathrm{ENC}}$ whenever it successfully convinces the clients.

Putting everything together, we arrive at the following theorem.

Theorem 1.3.6 (Informal)**.** *There exists an on-the-fly MPC protocol in the CRS-model with the following properties:*

- *Achieves security against malicious corruptions of an arbitrary subset of clients and possibly the server, under the Ring-LWE assumption and the assumption that the public key in the (modified) NTRU cryptosystem [HPS98, SS11b] is pseudorandom for a special setting of parameters.*
- *The offline phase runs in one (asynchronous) round of unidirectional communication from the parties to the server. The online phase runs in 5 rounds.*
- *The communication complexity of the online phase and the computation time of the computing parties therein is polylogarithmic in the size of the computation and the total size of the inputs, and linear in the size of their own input and the size of the output.*
- *The computation time of the server is polynomial in the size of the circuit.*

1.4 Related Work

We briefly survey related works in the areas of fully homomorphic encryption, MPC from homomorphic encryption, and MPC with the aid of a "cloud" server.

1.4.1 Fully Homomorphic Encryption

The notion of fully homomorphic encryption was first proposed by Rivest, Adleman, and Dertouzos [RAD78], but was only recently constructed in the groundbreaking result of Gentry [Gen09b, Gen09a]. In subsequent years, many improvements and new constructions have appeared in the literature [vDGHV10, BV11b, BV11a, BGV12, Bra12, BLLN13, GSW13, BV14].

Gentry's first construction [Gen09b, Gen09a] followed the following blueprint: first, he constructed a *somewhat homomorphic* encryption scheme working over ideal lattices, that was able to perform a limited number of evaluations. He then proved a *bootstrapping theorem*, showing that if a somewhat homomorphic scheme can homomorphically evaluate its own decryption circuit, then it can be converted into a *fully homomorphic* scheme. Unfortunately, Gentry's somewhat homomorphic scheme cannot evaluate its own decryption circuit and is therefore not bootstrappable. Nevertheless, he was able to construct a boostrappable scheme by *squashing* the decryption circuit sufficiently for the scheme to be able to homorphically evaluate it. Using this squashing technique required making an additional security assumption, namely, the sparse subset sum (SSS) assumption.

van Dijk et al. [vDGHV10] subsequently showed how to construct FHE over the integers, and Brakerski and Vaikuntanathan [BV11b] showed how to construct FHE from the Ring-LWE assumption of Lyubashevsky, Regev, and Peikert [LPR10]. Both of these works use squashing and bootstrapping, as in Gentry's original blueprint.

Gentry and Halevi [GH11a] showed how to use depth-3 arithmetic circuits and a hybrid of somewhat homomorphic encryption and multiplicatively homomorphic encryption (e.g. ElGamal encryption [Gam84]) to construct FHE without the use of squashing, and therefore without assuming the hardness of the SSS problem. In a separate work, Brakerski and Vaikuntanathan showed how to construct FHE from Regev's (standard) LWE assumption [Reg05, Reg09]. In this work, they introduced the techniques of *relinearization* and *modulus reduction*, which allowed them to forgo squashing as well. Gentry, Brakerski, and Vaikuntanathan [BGV12] later showed a refinement of these techniques into so-called *key-switching* and *modulus switching*, and showed how to build "leveled" homomorphic schemes that can evaluate circuits of *any a-priori known depth* without the use of squashing or bootstrapping. Formally, they show that for every $D \in \mathbb{N}$, there exists a homomorphic scheme $\mathcal{E}^{(D)}$ that is able to homomorphically evaluate circuits of depth D. Their technique involves switching to a smaller modulus after every level in a homomorphic computation, therefore requiring a fairly large modulus at the start of the computation. This required basing security of their scheme on the hardness of solving approximate-SVP to within sub-exponential factors. Coron et al. [CNT12] show how to apply the modulus reduction technique over the integers.

In work subsequent to ours, Brakerski [Bra12] showed a new noise-management technique that forwent the modulus switching step, allowing the use of a single modulus that is much smaller than the one needed in the BGV scheme. Security of Brakerski's scheme can be based on the hardness of solving approximate-SVP to within quasi-polynomial factors, a much weaker assumption. Bos et al. [BLLN13] show how to apply Brakerki's noise-management technique to the (multikey) FHE described in this dissertation [LTV12], based on the NTRU encryption scheme of Hofftein, Pipher, and Silverman [HPS98], with the modifications of Stehlé and Steinfeld [SS11b]. They further show that using these techniques, one can base security of the resulting FHE scheme on the Ring-LWE assumption alone, by using Stehlé and Steinfeld's original analysis. Their construction, however, is multikey for only a constant number of keys, which we show is an inherent property of any FHE scheme. Coron et al. [CLT14] show how to apply Brakerski's techniques over the integers.

Finally, Gentry et al. [GSW13] show how to construct a leveled homomorphic scheme that does not require the use an evaluation key to perform homomorphic computation, as do all previous schemes. Brakerski and Vaikuntanathan [BV14] show how to leverage the techniques of Gentry et al. [GSW13] to build a leveled homomorphic scheme that is as secure as standard (non-FHE) LWE-based public-key encryption.

Many other works study the efficiency of the schemes described above and present several optimizations [SV10, SS11a, GH11b, CMNT11, GHPS12, GHS12a, GHS12b, GHS12c, CCK$^+$13, SV14].

1.4.2 MPC from Homomorphic Encryption

The basic idea of using threshold homomorphic encryption (e.g. Paillier encryption [Pai99]) to boost the efficiency of MPC protocols was first presented by Cramer, Damgård, and Nielsen [CDN01], predating the existence of fully homomorphic encryption (first showed by Gentry in 2009 [Gen09b, Gen09a]). They show that if the parties have access to a public key for an additively homomorphic encryption scheme, and if they also have a corresponding secret key secret-shared among them, then they can evaluate any Boolean circuit "under the covers" of the encryption. Using the homomorphic properties of the scheme, the parties can locally evaluate all addition gates. Cramer et al. additionally show a short, interactive subprotocol for evaluating multiplica-

tion gates. After showing the first construction of fully homomorphic encryption, Gentry used the same template to show a generic MPC construction from any FHE [Gen09a].

In a work concurrent to ours, Myers, Sergi, and Shelat [MSS13] show a *black-box* construction of MPC from any threshold FHE scheme. Their main hurdle is devising a way for parties to prove plaintext knowledge of a ciphertext. To this end, they present a 2-round protocol for proving plaintext knowledge, which they construct from any circuit-private FHE scheme. Their protocol is not zero-knowledge [GO94], but it conserves the semantic security of the ciphertext in question. They also show how to construct threshold FHE using the scheme of van Dijk et al. [vDGHV10] over the integers. While the communication of their protocol is independent of the circuit-size of the function being computed, their protocol is not computation-efficient: parties compute proportional to the complexity of the function.

Other works by Damgård et al. [BDOZ11, DPSZ12, DKL$^+$13] build MPC from "semi-homomorphic" and somewhat homomorphic encryption. Their protocols require all parties to compute proportional to the complexity of the function at hand, and require interaction between parties at every gate. However, they display very good concrete efficiency. A work of Choudhury et al. [CLO$^+$13] shows how to trade computation efficiency for communication efficiency. Their protocol is parametrized by an integer L. Setting $L = 2$ yields a classic MPC protocol, in which interaction is required for computing every gate. As L increases, interaction is required less frequently, and only to "refresh" the computation after an increasing number of steps. Thus, at their heart of their construction lies an interactive "bootstrapping" protocol that refreshes ciphertexts during the evaluation.

Finally, a recent work by Garg et al. [GGHR14] shows how to achieve 2-round MPC in the CRS model from indistinguishability obfuscation ($i\mathcal{O}$) [BGI$^+$12]. As an optimization, they use multikey FHE (as defined in this work) to construct 2-round MPC with communication complexity that is independent of the circuit being computed. Though an efficient construction of $i\mathcal{O}$ is known for all circuits [GGH$^+$13b], its security is based on assumptions on multilinear maps [GGH13a] that are not very well understood yet.

1.4.3 MPC on the Cloud

The idea of using a powerful cloud server to alleviate the computational efforts of parties in an MPC protocol was recently explored in the work on "server-aided MPC" by Kamara, Mohassel, and Raykova [KMR11]. Their protocols, however, require some of the parties to do a large amount of work, essentially proportional to the size of the computation.

Halevi, Lindell, and Pinkas [HLP11] recently considered the model of "secure computation on the web" wherein the goal is to minimize interaction between the parties. While their definition requires absolutely no interaction among the participants of the protocol (they only interact with the server), they show that this notion can only be achieved for a small class of functions. Our goal, on the other hand, is to construct MPC protocols for *arbitrary functions*.

1.5 Roadmap

We have given a high-level overview of the results presented in this dissertation. Detailed descriptions of all the results highlighted in this introduction can be found in the corresponding chapters. In Chapter 2 we present preliminaries, definitions and technical tools used throughout the remaining chapters. Chapter 3 contains the definitions of cloud-assisted MPC and on-the-fly MPC, and

gives an overview of their constructions. Chapter 4 contains our construction of cloud-assisted MPC. Finally, in Chapter 5, we define multikey FHE and describe several constructions, and in Chapter 6 we show how to construct on-the-fly MPC from multikey FHE.

Chapter 2
Definitions and Preliminaries

2.1 Notation

In this work, we use the following notation. We use κ to denote the security parameter. For an integer n, we use the notation $[n]$ to denote the set $[n] \stackrel{\mathsf{def}}{=} \{1, \ldots, n\}$. For a randomized function f, we write $f(x; r)$ to denote the unique output of f on input x with random coins r. We write $f(x)$ to denote a random variable for the output of $f(x; r)$ over uniformly random coins r. For a distribution or random variable X, we write $x \leftarrow X$ to denote the operation of sampling a random x according to X. For a set S, we overload notation and use $s \leftarrow S$ to denote sampling s from the uniform distribution over S. We use $y := f(x)$ to denote the deterministic evaluation of f on input x with output y. For two distributions, X and Y, we use $X \stackrel{c}{\approx} Y$ to mean that X and Y are computationally indistinguishable, and $X \stackrel{s}{\approx} Y$ to mean that they are statistically close.

2.2 Σ-Protocols and Zero-Knowledge Proofs

Σ-Protocols. We recall the notion of *gap* Σ-protocols [AJW11], a weaker version of Σ-protocols [CDS94], where honest-verifier zero-knowledge holds for all statements in some NP relation R_{zk} but soundness only holds w.r.t. $R_{\mathsf{sound}} \supseteq R_{\mathsf{zk}}$. In other words, zero-knowledge is guaranteed for an honest prover holding a statement in R_{zk}, but an honest verifier is only convinced that the statement is in a larger set $R_{\mathsf{sound}} \supseteq R_{\mathsf{zk}}$.

Definition 2.2.1 (Gap Σ-Protocol)**.** *Let R_{zk} and R_{sound} be two NP relations such that $R_{\mathsf{zk}} \subseteq R_{\mathsf{sound}} \subseteq \{0,1\}^* \times \{0,1\}^*$, and let L_{zk} and L_{sound} be their corresponding NP languages. A* gap Σ-protocol *for $(R_{\mathsf{zk}}, R_{\mathsf{sound}})$ is a 3-step interactive protocol $\langle P, V \rangle$ between a prover $P = (P_1, P_2)$ and a verifier $V = (V_1, V_2)$, with the following syntax:*

- *$(a, st) \leftarrow P_1(x, w)$: Given a statement and witness pair (x, w), outputs a message a and a state string st.*
- *$c \leftarrow V_1(x, a)$: Given a statement x and message a, outputs a random challenge c from a challenge space $\mathcal{C}$.*
- *$z \leftarrow P_2(st, c)$: Given a state string st and a challenge c, outputs an answer z.*
- *$b \leftarrow V_2(x, a, c, z)$: Given a statement x, a message a, a challenge c, and an answer z, outputs a bit b, i.e. either accepts or rejects the transcript (a, c, z) for statement x.*

We require that the following three properties hold:

Completeness: *For any* $(x, w) \in R_{\mathsf{zk}}$,

$$\Pr\left[\begin{array}{c|c} V_2(x,a,c,z) = 1 & \begin{array}{c}(a, st) \leftarrow P_1(x, w)\\ c \leftarrow V_1(x, a)\\ z \leftarrow P_2(st, c)\end{array}\end{array}\right] = 1$$

Special Soundness: *There exists an "extractor" such that for any two accepting transcripts* (a, c, z) *and* (a, c', z') *for the same statement* x *with* $c \neq c'$*, the extractor outputs a valid witness for* $x \in R_{\mathsf{sound}}$*. Formally, there exists a* PPT *algorithm* Ext *such that for all* x *and all* (a, c, z) *and* (a, c', z') *such that* $c \neq c'$ *and* $V_2(x, a, c, z) = V_2(x, a, c', z') = 1$*:*

$$\Pr\left[\ (x, w) \notin R_{\mathsf{sound}} \ \middle|\ w \leftarrow \mathsf{Ext}(x, a, c, z, c', z')\ \right] = 1$$

Honest-Verifier Zero Knowledge (HVZK): *There exists a* PPT *simulator* Sim *that "simulates" valid transcripts without knowing a witness, if it sees the challenge beforehand. Formally, there exists* PPT *algorithm* Sim *such that for all* $(x, w) \in R_{\mathsf{zk}}$ *and all* $c \in \mathcal{C}$*, we have:*

$$\left[\ (a, c, z)\ \middle|\ \begin{array}{c}(a, st) \leftarrow P_1(x, w)\\ z \leftarrow P_2(st, c)\end{array}\ \right] \overset{s}{\approx} \left[\ (a', c, z')\ \middle|\ (a', z') \leftarrow \mathsf{Sim}(x, c)\ \right]$$

For an NP relation R with corresponding language L, a well-known construction using Σ-protocols allows a prover to show that either $x_0 \in L$ or $x_1 \in L$ *without revealing which one holds.* Suppose $\langle P, V\rangle$ is a Σ-protocol for R; we construct a new protocol for proving that either $x_0 \in L$ or $x_1 \in L$. Let b be such that $(x_b, w_b) \in R$ for some witness w_b known to the prover. The prover chooses c_{1-b} at random from the challenge space $\mathcal{C}$ and runs $(a_b, st) \leftarrow P_1(x_b, w_b)$ and $(a_{1-b}, z_{1-b}) \leftarrow \mathsf{Sim}(x, c_{1-b})$. It sends (a_0, a_1) to the verifier, who returns a challenge c. The prover computes $c_b = c - c_{1-b}$, runs $z_b \leftarrow P_2(st, c)$ and sends (c_0, c_1, z_0, z_1) to the verifier, who checks that $V_2(x_0, a_0, c_0, z_0) = V_2(x_1, a_1, c_1, z_1) = 1$ and $c = c_0 + c_1$. The resulting protocol is called an *OR* Σ*-protocol.* The theorem below modifies this to the setting of *gap* Σ-protocols.

Theorem 2.2.1. *Let* R_{zk} *and* R_{sound} *be two NP relations such that* $R_{\mathsf{zk}} \subseteq R_{\mathsf{sound}} \subseteq \{0,1\}^* \times \{0,1\}^*$*, and let* $\langle P, V\rangle$ *be a gap* Σ*-protocol for* $(R_{\mathsf{zk}}, R_{\mathsf{sound}})$*. The construction described above is a gap OR* Σ*-protocol for* $(R_{\mathsf{zk}}, R_{\mathsf{sound}})$*.*

Non-Interactive Zero-Knowledge (NIZK). We also recall the notion of *non-interactive zero-knowledge (NIZK)* [BFM88]. For our purposes, it is more convenient to use the notion of *(same-string) NIZK arguments* from [SCO+01]. This definition and all our constructions that use it can be extended in the natural way to NIZK proofs, where soundness holds for all unbounded adversaries[1].

Definition 2.2.2 (NIZK)**.** *Let* R *be an NP relation on pairs* (x, w) *with corresponding language* $L = \{x \mid \exists\ w\ s.t.\ (x, w) \in R\}$*. A* non-interactive zero-knowledge (NIZK) argument system *for* R *consists of three algorithms* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *with syntax:*

- $(\mathsf{crs}, \mathsf{tk}) \leftarrow \mathsf{Setup}(1^\kappa)$*: Outputs a common reference string (CRS)* crs *and a trapdoor key* tk *to the CRS.*

[1] Apart from modifying the soundness condition, in the setting of proofs key generation samples a CRS but not a trapdoor, and the zero-knowledge simulator first samples a *simulated CRS* that is computationally indistinguishable from the real CRS, and a trapdoor to this CRS.

- $\pi \leftarrow \mathsf{Prove}_{\mathsf{crs}}(x, w)$*: Outputs an argument* π *showing that* $R(x, w) = 1$.
- $0/1 \leftarrow \mathsf{Verify}_{\mathsf{crs}}(x, \pi)$*: Verifies whether or not the argument* π *is correct.*

For the sake of clarity, we write Prove *and* Verify *without the* crs *in the subscript when the* crs *can be inferred from context. We require that the following three properties hold:*

Completeness: *For any* $(x, w) \in R$,

$$\Pr\left[\ \mathsf{Verify}(x,\pi) = 1 \ \middle|\ \begin{array}{c}(\mathsf{crs},\mathsf{tk}) \leftarrow \mathsf{Setup}(1^\kappa)\\ \pi \leftarrow \mathsf{Prove}(x,w)\end{array}\ \right] = 1$$

Soundness: *For any* PPT *adversary* $\widetilde{P}$,

$$\Pr\left[\ \begin{array}{c}\mathsf{Verify}(x^*,\pi^*) = 1\\ x^* \notin L\end{array}\ \middle|\ \begin{array}{c}(\mathsf{crs},\mathsf{tk}) \leftarrow \mathsf{Setup}(1^\kappa)\\ (x^*,\pi^*) \leftarrow \widetilde{P}(\mathsf{crs})\end{array}\ \right] = \mathsf{negl}(\kappa).$$

Unbounded Zero-Knowledge: *There exists a* PPT *simulator* Sim *that "simulates" valid proofs without knowing a witness, but with the aid of the trapdoor key. We start by defining two oracles.*

The Prover Oracle: A query to the prover oracle $\mathcal{P}(\cdot)$ *consists of a pair* (x, w)*. The oracle checks if* $(x, w) \in R$*. If so, it outputs a valid argument* $\mathsf{Prove}(x, w)$*; otherwise it outputs* $\perp$.

The Simulation Oracle: A query to the simulation oracle $\mathcal{SIM}_{\mathsf{tk}}(\cdot)$ *consists of a pair* (x, w)*. The oracle checks if* $(x, w) \in R$*. If so, it ignores* w *and outputs a simulated argument* $\mathsf{Sim}(\mathsf{tk}, x)$*; otherwise it outputs* $\perp$.

Formally, we require that for any PPT *adversary* $\mathcal{A}$*, the advantage of* $\mathcal{A}$ *in the following game is negligible (in* κ*):*

- *The challenger samples* $(\mathsf{crs}, \mathsf{tk}) \leftarrow \mathsf{Setup}(1^\kappa)$ *and gives* crs *to* $\mathcal{A}$*. The challenger also samples a bit* $b \leftarrow \{0, 1\}$.
- *If* $b = 0$*, the adversary* $\mathcal{A}$ *is given access to the prover oracle* $\mathcal{P}(\cdot)$*. If* $b = 1$*,* $\mathcal{A}$ *is given access to the simulation oracle* $\mathcal{SIM}_{\mathsf{tk}}(\cdot)$*. In either case, the adversary can adaptively access its oracle..*
- *The adversary* $\mathcal{A}$ *outputs a bit* $\tilde{b}$.

The advtange *of* $\mathcal{A}$ *is defined to be* $\left|\Pr[\tilde{b} = b] - \frac{1}{2}\right|$.

Fiat and Shamir [FS86] showed how to convert a Σ protocol $\langle P, V\rangle$ for an NP relation R into a NIZK argument for R secure in the random oracle model [BR93]. Informally, the CRS contains a description of a hash function H, which is modeled as a random oracle. To compute a non-interactive argument, the prover runs $(a, st) \leftarrow P_1(x, w)$ and obtains the verifier's challenge by applying the hash function to a and x: $c := H(a, x)$. It then computes $z \leftarrow P_2(st, c)$ and sends the argument $\pi = (a, c, z)$. The verifier runs $V_2(x, a, c, z)$ to verify the argument. The theorem below modifies this to the setting of *gap* Σ-protocols.

Theorem 2.2.2 ([FS86]). *Let R_{zk} and R_{sound} be two NP relations such that $R_{\mathsf{zk}} \subseteq R_{\mathsf{sound}} \subseteq \{0,1\}^* \times \{0,1\}^*$, and let $\langle P, V\rangle$ be a gap Σ-protocol for $(R_{\mathsf{zk}}, R_{\mathsf{sound}})$. Applying the Fiat-Shamir transform to $\langle P, V\rangle$ yields a non-interactive zero-knowledge (NIZK) argument system where soundness holds w.r.t. R_{sound} and completeness and zero-knowledge hold w.r.t. R_{zk}, secure in the random oracle model.*

Though secure in the random oracle model, we remark that in some cases standard-model security of the resulting NIZK appears to be harder to achieve, as we prove in an independent work [DJKL12, BDG+13]. In particular, we show that if the language L is quasi-polynomially hard and the protocol has messages of size $\mathsf{polylog}(\kappa)$ and is $\kappa^{\log \kappa}$-HVZK, then the resulting NIZK cannot be proven sound via a black-box reduction to a (super-polynomially hard) falsifiable assumption [Nao03].

2.3 Succinct Non-Interactive Arguments: SNARGs and SNARKs

We review the definitions of succinct non-interactive arguments (SNARGs) and succinct non-interactive arguments of knowledge (SNARKs); we use the formalization of Gentry and Wichs [GW11], and Bitansky et al. [BCCT12]. As in the work of Bitansky et al., we allow the proof size to be polynomial in the size of the statement, but require it to be polylogarithmic in the size of the witness. We also require fast proof verification.

Definition 2.3.1 (SNARG). *Let R be an NP relation on pairs (x, w) with corresponding language $L = \{x \mid \exists\ w \text{ s.t. } (x, w) \in R\}$. A* succinct non-interactive argument (SNARG) system *for L consists of three algorithms $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ with syntax:*

- $(\mathsf{vrs}, \mathsf{priv}) \leftarrow \mathsf{Setup}(1^\kappa)$*: Outputs a verification reference string* vrs *and a private verification state* priv*.*
- $\varphi \leftarrow \mathsf{Prove}(\mathsf{vrs}, x, w)$*: Outputs an argument φ showing that $R(x, w) = 1$.*
- $0/1 \leftarrow \mathsf{Verify}(\mathsf{priv}, x, \varphi)$*: Verifies whether or not the argument φ is correct.*

We require that the following properties hold:

Completeness: *For any $(x, w) \in R$,*

$$\Pr\left[\ \mathsf{Verify}(\mathsf{priv}, x, \varphi) = 1 \ \middle|\ \begin{array}{c}(\mathsf{vrs}, \mathsf{priv}) \leftarrow \mathsf{Setup}(1^\kappa)\\ \varphi \leftarrow \mathsf{Prove}(\mathsf{vrs}, x, w)\end{array}\ \right] = 1$$

In addition, $\mathsf{Prove}(\mathsf{vrs}, x, w)$ runs in time $\mathsf{poly}(\kappa, |x|, |w|)$.

Adaptive Soundness: *For any* PPT *adversary $\widetilde{P}$,*

$$\Pr\left[\ \begin{array}{c}\mathsf{Verify}(\mathsf{priv}, x^*, \varphi^*) = 1 \ \wedge\\ x^* \notin L\end{array}\ \middle|\ \begin{array}{c}(\mathsf{vrs}, \mathsf{priv}) \leftarrow \mathsf{Setup}(1^\kappa)\\ (x^*, \varphi^*) \leftarrow \widetilde{P}(\mathsf{vrs})\end{array}\ \right] = \mathsf{negl}(\kappa).$$

Succinctness: *The length of the proof and the time required for its verification are polylogarithmic in the size of the witness, i.e.* $\mathsf{poly}(\kappa)\,(\mathsf{poly}(|x|) + \mathsf{polylog}(|w|))$.

Definition 2.3.2 (SNARK). *A SNARG* $\Phi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *is additionally a proof of knowledge, or a* succinct non-interactive argument of knowledge (SNARK) *if it satisfies the following stronger definition of soundness:*

Adaptive Extractability: *There exists an extractor* Ext *that "extracts" a valid witness from any valid proof* φ*. Formally, for any* PPT *adversary* $\widetilde{P}$*, there exists a* PPT *algorithm* Ext *such that:*

$$\Pr\left[\begin{array}{c|c} \mathsf{Verify}(\mathsf{priv}, x^*, \varphi^*) = 1 \wedge & (\mathsf{vrs}, \mathsf{priv}) \leftarrow \mathsf{Setup}(1^\kappa) \\ R(x^*, w') = 0 & (x^*, \varphi^*) \leftarrow \widetilde{P}(\mathsf{vrs}) \\ & w' \leftarrow \mathsf{Ext}(x^*, \varphi^*) \end{array}\right] = \mathsf{negl}(\kappa)$$

Public vs. Private Verifiability. In the case where $\mathsf{priv} = \mathsf{vrs}$, we say that the SNARG or SNARK is *publicly verifiable*. In this case, anyone can verify all proofs. Otherwise, we say that it is a *designated-verifier* SNARG/SNARK, in which case soundness/extractability is only guaranteed as long as priv remains secret to the prover. In this case, only the party holding priv can verify the proof.

2.3.1 Delegation of Computation from SNARGs

In delegation of computation we are concerned with at a client C, who wishes to delegate the computation of a pre-specified polynomial-time algorithm M on an input x, to a worker W. The client additionally wishes to verify the correctness of the output y returned by W (i.e. verify that $y = M(x)$) in time that is significantly smaller than the time required to compute $M(x)$ from scratch.

SNARGs can be used in this setting as follows: Define the NP language: $L_M = \{\ (x, y)$ such that $M(x) = y\ \}$. A straight-forward witness to the statement $(x, y) \in L_M$ consists of the steps taken by M in a computation of $M(x)$ resulting in the output y. The size of this witness is proportional to the size of the computation. Using a SNARG guarantees that the size of the proof is polylogaritmic in the size of the witness, and therefore polylogarithmic in the size of the computation.

2.3.2 Constructions

Gentry and Wichs [GW11] proved that standard-model security of SNARGs with adaptive soundness and proof size sublinear in the witness and statement sizes, cannot be based on any falsifiable assumption [Nao03]. The constructions we show below either assume a random oracle [BR93] or most often use a non-falsifiable assumption.

CS Proofs. Kilian [Kil92, Kil95] showed how to perform succinct *interactive* verification for any NP language. His solution describes a 4-round protocol, where the prover first constructs a PCP for the correctness of the computation and then uses Merkle hashes to compress it to a sufficiently small proof. Micali's CS proofs [Mic94] apply the Fiat-Shamir transform [FS86] to Kilian's protocol, obtaining a non-interactive solution. CS proofs are publicly verifiable SNARGs (and SNARKs under Valiant's analysis [Val08]); indeed, the only "setup" required is a description of a hash function H to use as the random oracle. This can be ensured by letting the vrs be a random key for a (say) collision-resistant hash function.

Due to its use of the Fiat-Shamir transform, Micali's solution is only secure in the *random oracle model* [BR93]. Unfortunately, several results have shown the implausibility of instantiating

the random oracle in the Fiat-Shamir transform with any explicit hash function [HT98, Bar01, CGH04, DNRS03, GK03]. In particular, in an independent work [DJKL12, BDG+13], we show that the security of CS proofs (even with non-adaptive soundness) cannot be based on any falsifiable assumption. On the other hand, it has been shown that the security of the Fiat-Shamir paradigm can be based on specific non-falsifiable assumptions regarding the existence of robust randomness condensers for seed-dependent sources [BLV06, DRV12].

Constructions will Small VRS. Bitansky et al. [BCCT12, BCCT13] and Goldwasser et al. [GLR11] revisit the construction of CS proofs and, based on the works of Di Crescenzo and Lipmaa [CL08] and Valiant [Val08], show how to construct SNARGs and SNARKs based on a different non-falsifiable assumption relating to the existence of extractable collision-resistant hash functions. In these works, the verifier's entire computation (both in computing its reference string vrs and in verifying the proof) depends only polylogarithmically in the size of the witness (i.e. the delegated computation). The SNARGs and SNARKs in these works are designated-verifier.

Allowing a Large VRS. Another series of works [Gro10, Lip12, GGPR13, PHGR13, Lip13] constructs SNARGs and SNARKs where the verifier's reference string vrs is allowed to depend on the circuit being delegated. In particular, Groth's construction [Gro10] has a VRS of size quadratic in the circuit size. Lipmaa [Lip12] reduces this size to be quasi-linear, and the works of Gennaro et al. [GGPR13] and Parno et al. [PHGR13] further reduce it to linear in the circuit size. Lipmaa [Lip13] refines the construction of Gennaro et al. to reduce the magnitude of the constant in the size of the VRS. All of these constructions are based on certain number-theoretic non-falsifiable assumptions.

2.4 Secure Multiparty Computation (MPC)

Let f be an N-input function with single output. A *multiparty protocol* Π for f is a protocol between N interactive Turing Machines $P_1, \ldots, P_N$, called *parties*, such that for all $\vec{x} = (x_1, \ldots, x_N)$, the output of Π in an execution where P_i is given x_i as input, is $y \stackrel{\mathsf{def}}{=} f(\vec{x})$.

2.4.1 Security in the Ideal/Real Paradigm

Informally, a multiparty protocol Π is *secure* if after running Π, no colluding set of corrupt parties can learn anything about an honest player's input or change the output of an honest party. We formalize this in the Ideal/Real paradigm (see e.g. [Gol04]).

Ideal and Real Worlds. We define an *ideal world* in which the computation of f is performed through a trusted functionality $\mathcal{F}$ that receives inputs x_i from each party P_i, computes $y \stackrel{\mathsf{def}}{=} f(x_1, \ldots, x_N)$ and gives y to all parties $P_1, \ldots, P_N$. It is clear that in the ideal world, the only information that any party learns is its own input and the output y. We also define a *real world* in which parties $P_1, \ldots, P_N$ run the protocol Π.

The Network. We assume that the real-world execution of the protocol is performed over a *secure* and *synchronous* network; that is, we assume that parties can reliably send messages to other parties

without these being read or altered in transmission, and that all point-to-point communications happen at the same time. We also assume that a secure *broadcast* channel is available to all parties.

The Adversary. In either world, we consider a single adversary that is allowed to corrupt any subset of $t < N$ parties. An adversary is modeled as an interactive Turing Machine that receives all messages directed to the corrupted parties and controls the messages sent by them. In this work, we consider only *static* adversaries, that is, adversaries that select the subset of corrupted parties non-adaptively, before any computation is performed. On the other hand, we assume that in each round of the protocol, the adversary chooses the messages for the corrupted parties adaptively, based on the entire transcript of the protocol, up to that round.

We remark that our results can be extended to achieve security against *rushing* real-world adversaries who, on any given round, choose the messages for the corrupted parties adaptively, based on the entire transcript of the protocol *and the messages of the honest parties on that round*. Note that rushing adversaries correspond to a semi-synchronous model of communication.

Output Distributions. We use $\text{IDEAL}_{\mathcal{F},\mathcal{S}}(\vec{x})$ to denote the joint output of an ideal-world adversary $\mathcal{S}$ and parties $P_1, \ldots, P_N$ in an ideal execution with functionality $\mathcal{F}$ and inputs $\vec{x} = (x_1, \ldots, x_N)$. Similarly, we use $\text{REAL}_{\Pi,\mathcal{A}}(\vec{x})$ to denote the joint output of a real-world adversary $\mathcal{A}$ and parties $P_1, \ldots, P_N$ in an execution of protocol Π with inputs $\vec{x} = (x_1, \ldots, x_N)$.

We say that a protocol Π *securely realizes* $\mathcal{F}$ *against the class of adversaries* Adv, if for every real-world adversary $\mathcal{A} \in \mathsf{Adv}$, there exists an ideal-world adversary $\mathcal{S}$ with black-box access to $\mathcal{A}$ such that for all input vectors $\vec{x}$,

$$\text{IDEAL}_{\mathcal{F},\mathcal{S}}(\vec{x}) \overset{c}{\approx} \text{REAL}_{\Pi,\mathcal{A}}(\vec{x})$$

2.4.2 Types of Adversaries

As stated above, in this work we only consider classes of adversaries Adv containing static adversaries that corrupt any subset of $t < N$ parties. We now describe three different types of adversaries: malicious, semi-honest, and semi-malicious. The first two are used extensively in the literature, while the latter was introduced recently by Asharov et al. [AJW11, AJL+12]. Of these, malicious adversaries are the strongest, and it is our end goal to achieve security against them in all our protocols.

It is customary to prove security against semi-honest adversaries as a stepping stone to proving security against malicious adversaries. However, in this work we follow a different path and first prove security against semi-malicious adversaries. We then show how to modify the protocol at hand to achieve security against malicious adversaries. For completeness, we describe all three types of adversaries below and describe how security against one type is related to security against another.

Semi-Honest Adversaries. A semi-honest adversary, also known as an honest-but-curious adversary, is one that follows the protocol as described (samples randomness from the correct distribution, and computes the specified message at each round), but given its view of the protocol will try to learn information about honest players' inputs.

Malicious Adversaries. A malicious adversary is not restricted in how it samples random elements or how it computes its messages at each round. It can sample random elements from any arbitrary distribution, and compute the messages of corrupted parties in any arbitrary way, adaptively, according to the partial view it has seen up to that point.

Semi-Malicious Adversaries. Recall that an adversary is modeled as an interactive Turing Machine (ITM). A semi-malicious adversary is an ITM with an additional *witness tape*. At each round ℓ and for every corrupted party P_j, the adversary must write on the special witness tape, *some* witness pair $\left(x_j^{(\ell)}, r_j^{(\ell)}\right)$ of input and randomness that explains the message $m_j^{(\ell)}$ sent by P_j on that round. More formally, the messages of a corrupted party P_j must match those of the specified honest protocol when at each round ℓ party P_j is run with input and randomness $\left(x_j^{(\ell)}, r_j^{(\ell)}\right)$.

A semi-malicious adversary can sample random elements from any arbitrary distribution, but it must follow the correct behavior of the honest protocol with inputs and randomness that it *knows*. It is therefore weaker than a malicious adversary, who might not know witnesses for the messages it sends at every round, but stronger than a semi-honest adversary, whose witnesses at every round are distributed honestly.

From Semi-Malicious to Malicious Security. Asharov et al. [AJW11, AJL$^+$12] show how to generically transform a protocol that is secure against semi-malicious adversaries into one that is secure against malicious adversaries. The idea behind the compiler is to have each party prove in zero-knowledge that every message it sends follows the honest protocol and is consistent with all previous messages. In particular, this forces all parties to know witnesses that explain their behavior at every round. The same compiler works in our security model with one subtlety: instead of using standard zero-knowledge proofs, the protocol must use *zero-knowledge proofs of knowledge*. This is to ensure that the simulator can extract the witness $w_j^{(\ell)}$ from the proof sent on round ℓ by the malicious adversary on behalf of the corrupted party P_j. We refer the reader to the work of Asharov et al. [AJW11, AJL$^+$12] for more details.

Finally, we note that unlike the standard GMW compiler from semi-honest security to malicious security [GMW87], the parties are not required to perform any coin-flipping. This, in particular, reduces the round complexity of the resulting protocol.

2.5 Fully Homomorphic Encryption (FHE)

We review the definitions of fully and leveled homomorphic encryption.

Definition 2.5.1 ($\mathcal{C}$-Homomorphic Encryption [Gen09b])**.** *For a class of circuits $\mathcal{C}$, a $\mathcal{C}$-homomorphic encryption scheme is a tuple of algorithms $\mathcal{E} = (\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ with the following syntax:*

- $\mathsf{params} \leftarrow \mathsf{Setup}(1^\kappa)$*: For security parameter κ, outputs public parameters* params*. All other algorithms,* $\mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval}$*, implicitly take* params *as input, even when not explicitly stated.*
- $(\mathsf{pk}, \mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{Keygen}(1^\kappa)$*: For a security parameter κ, outpus a public key* pk*, a secret key* sk*, and a (public) evaluation key* ek*.*

- $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$*: Given a public key* pk *and a message* m*, outputs a ciphertext* c*.*
- $m := \mathsf{Dec}(\mathsf{sk}, c)$*: Given a secret key* sk *and a ciphertext* c*, outputs a message* m*.*
- $c := \mathsf{Eval}(\mathsf{ek}, C, c_1, \ldots, c_\ell)$*: Given an evaluation key* ek*, a (description of a) circuit* C *and* ℓ *ciphertexts* $c_1, \ldots, c_\ell$*, outputs a ciphertext* c*.*

We require that for all $c \in \mathcal{C}$*, all* $(\mathsf{pk}, \mathsf{sk}, \mathsf{ek})$ *in the support of* $\mathsf{Keygen}(1^\kappa)$ *and all plaintexts* $(m_1, \ldots, m_\ell)$ *and ciphertexts* $(c_1, \ldots, c_\ell)$ *such that* c_i *is in the support of* $\mathsf{Enc}(\mathsf{pk}, m_i)$*, if* $c := \mathsf{Eval}(\mathsf{ek}, C, c_1, \ldots, c_\ell)$*, then* $\mathsf{Dec}(\mathsf{sk}, c) = C(m_1, \ldots, m_\ell)$*.*

Definition 2.5.2 (Fully Homomorphic Encryption [Gen09b])**.** *An encryption scheme* $\mathcal{E}$ *is* fully homomorphic *if it satisfies the following properties:*

Correctness: $\mathcal{E}$ *is* $\mathcal{C}$*-homomorphic for the class* $\mathcal{C}$ *of all circuits.*

Compactness: *The computational complexity of* $\mathcal{E}$*'s algorithms is polynomial in the security parameter* κ*, and in the case of the evaluation algorithm, the size of the circuit.*

We now state the definition of leveled homomorphic encryption from [BGV12], which is a relaxation of the original definition of fully homomorphic encryption (Definition 2.5.2). The main difference is that Definition 2.5.2 requires all algorithms (decryption in particular) to be independent of the circuit(s) that the scheme can evaluate. Leveled homomorphic encryption relaxes this definition to let all algorithms (including decryption) depend on the circuit depth D.

Definition 2.5.3 (Leveled Homomorphic Encryption [BGV12])**.** *Let* $\mathcal{C}^{(D)}$ *be the class of all circuits of depth at most* D *(that use some specified complete set of gates). We say that a family of homomorphic encryption schemes* $\{\mathcal{E}^{(D)} : D \in \mathbb{Z}^+\}$ *is leveled fully homomorphic if, for all* $D \in \mathbb{Z}^+$*, it satisfies the following properties:*

Correctness: $\mathcal{E}^{(D)}$ *is* $\mathcal{C}^{(D)}$*-homomorphic.*

Compactness: *The computational complexity of* $\mathcal{E}^{(D)}$*'s algorithms is polynomial in the security parameter* κ *and* D*, and in the case of the evaluation algorithm, the size of the circuit. We emphasize that this polynomial must be the same for all* D*.*

2.5.1 Bootstrapping

We remind the reader of the definition of a bootstrappable encryption scheme and present Gentry's bootstrapping theorem [Gen09b, Gen09a] that states that a bootstrappable scheme can be converted into a fully homomorphic one.

Definition 2.5.4 (Bootstrappable Scheme)**.** *Let* $\mathcal{E} = (\mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *be a* $\mathcal{C}$*-homomorphic encryption scheme, and let* f_{add} *and* f_{mult} *be the augmented decryption functions of the scheme defined as*

$$f_{\mathsf{add}}^{c_1,c_2}(\mathsf{sk}_1, \ldots, \mathsf{sk}_N) = \mathsf{Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_N, c_1) \quad XOR \quad \mathsf{Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_N, c_2)$$

$$f_{\mathsf{mult}}^{c_1,c_2}(\mathsf{sk}_1, \ldots, \mathsf{sk}_N) = \mathsf{Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_N, c_1) \quad AND \quad \mathsf{Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_N, c_2)$$

$\mathcal{E}$ *is* bootstrappable *if* $\left\{f_{\mathsf{add}}^{c_1,c_2}, f_{\mathsf{mult}}^{c_1,c_2}\right\}_{c_1,c_2} \subseteq \mathcal{C}$*, namely, if it can homomorphically evaluate* f_{add} *and* f_{mult}*.*

Definition 2.5.5 (Weak Circular Security)**.** *A public-key encryption scheme $\mathcal{E} = (\mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec})$ is weakly circular secure if it is IND-CPA secure even for an adversary with auxiliary information containing encryptions of all secret key bits: $\{\mathsf{Enc}(\mathsf{pk}, \mathsf{sk}[i])\}_i$. Namely, no polynomial-time adversary can distinguish an encryption of 0 from an encryption of 1, even given this additional information.*

Theorem 2.5.1 (Bootstrapping Theorem)**.** *Let $\mathcal{E}$ be a bootstrappable scheme that is also weakly circular secure. Then there exists a fully homomorphic encryption scheme $\mathcal{E}'$.*

2.6 Rings

In this section we introduce preliminaries to our concrete constructions, which are all ring-based. Some of the discussion in this section is taken verbatim from the work of Brakerski and Vaikuntanathan [BV11b].

We work over rings $R \stackrel{\mathsf{def}}{=} \mathbb{Z}[x]/\langle\phi(x)\rangle$ and $R_q \stackrel{\mathsf{def}}{=} R/qR$ for some degree $n = n(\kappa)$ integer polynomial $\phi(x) \in \mathbb{Z}[x]$ and a prime integer $q = q(\kappa) \in \mathbb{Z}$. Note that R_q is isomorphic to $\mathbb{Z}_q[x]/\langle\phi(x)\rangle$, the ring of degree n polynomials modulo $\phi(x)$ with coefficients in $\mathbb{Z}_q$. Addition in these rings is done component-wise in their coefficients (thus, their additive group is isomorphic to $\mathbb{Z}^n$ and $\mathbb{Z}_q^n$ respectively), and multiplication is polynomial multiplication modulo $\phi(x)$ (and also q, in the case of the ring R_q). An element in R (or R_q) can be viewed as a degree $(n-1)$ polynomial over $\mathbb{Z}$ (or $\mathbb{Z}_q$). We represent such an element using the vector of its n coefficients. In the case of R_q each coefficient is in the range $\{-\lfloor\frac{q}{2}\rfloor, ..., \lfloor\frac{q}{2}\rfloor\}$. For an element $a(x) = a_0 + a_1 x + \ldots + a_{n-1}x^{n-1} \in R$, we let $\|a\|_\infty = \max|a_i|$ denote its ℓ_∞ norm.

In this work, we set $\phi(x) = x^n + 1$ where n is a power of two, and use distributions over the ring $R \stackrel{\mathsf{def}}{=} \mathbb{Z}[x]/\langle\phi(x)\rangle$. For the purpose of homomorphism, the only important property of these distributions is the magnitude of the coefficients of a polynomial output by the distribution. Hence, we define a B-bounded distribution to be a distribution over R where the ℓ_∞-norm of a sample is bounded by B.

Definition 2.6.1. (B-Bounded Polynomial) *A polynomial $e \in R$ is called* B-bounded *if $\|e\|_\infty \leq B$.*

Definition 2.6.2. (B-Bounded Distribution) *A distribution ensemble $\{\chi_\kappa\}_{\kappa\in\mathbb{N}}$, supported over R, is called* B-bounded *(for $B = B(\kappa)$) if for all e in the support of χ_κ, we have $\|e\|_\infty < B$. In other words, a B-bounded distribution over R outputs a B-bounded polynomial.*

The following lemma says that multiplication in the ring $\mathbb{Z}[x]/\langle x^n + 1\rangle$ increases the norm of the constituent elements only by a small amount.

Lemma 2.6.1. *Let $n \in \mathbb{N}$, let $\phi(x) = x^n + 1$ and let $R = \mathbb{Z}[x]/\langle\phi(x)\rangle$. For any $s, t \in R$,*

$$||s \cdot t|| \leq \sqrt{n} \cdot ||s|| \cdot ||t|| \qquad \text{and} \qquad ||s \cdot t||_\infty \leq n \cdot ||s||_\infty \cdot ||t||_\infty$$

Lemma 2.6.1 yields the following corollary.

Corollary 2.6.2. *Let $n \in \mathbb{N}$, let $\phi(x) = x^n + 1$ and $R = \mathbb{Z}[x]/\langle\phi(x)\rangle$. Let χ be a B-bounded distribution over the ring R and let $s_1, \ldots, s_k \leftarrow \chi$. Then $s \stackrel{\mathsf{def}}{=} \prod_{i=1}^k s_i$ is $(n^{k-1}B^k)$-bounded.*

2.6.1 Discrete Gaussians

For any real $r > 0$ the Gaussian function on $\mathbb{R}^n$ centered at $\mathbf{c}$ with parameter r is defined as:

$$\forall \mathbf{x} \in \mathbb{R}^n \ : \quad \rho_{r,\mathbf{c}}(\mathbf{x}) \stackrel{\mathsf{def}}{=} e^{-\pi \|\mathbf{x}-\mathbf{c}\|^2/r^2}$$

Definition 2.6.3. *For any $n \in \mathbb{N}$ and for any $\mathbf{c} \in \mathbb{R}^n$ and real $r > 0$, the* Discrete Gaussian distribution *over $\mathbb{Z}^n$ with standard deviation r and centered at $\mathbf{c}$ is defined as:*

$$\forall \mathbf{x} \in \mathbb{Z}^n \ : \quad D_{\mathbb{Z}^n,r,\mathbf{c}} \stackrel{\mathsf{def}}{=} \frac{\rho_{r,\mathbf{c}}(\mathbf{x})}{\rho_{r,\mathbf{c}}(\mathbb{Z}^n)}$$

where $\rho_{r,\mathbf{c}}(\mathbb{Z}^n) \stackrel{\mathsf{def}}{=} \sum_{\mathbf{x}\in\mathbb{Z}^n} \rho_{r,\mathbf{c}}(\mathbf{x})$ is a normalization factor.

We present some elementary facts about the Gaussian distribution. The first fact shows that the discrete Gaussian distribution over $\mathbb{Z}^n$ with standard deviation r outputs a $(r\sqrt{n})$-bounded polynomial with high probability. This allows us to define a *truncated* Gaussian distribution that is $(r\sqrt{n})$-bounded and statistically close to the discrete Gaussian.

Lemma 2.6.3 (MR07)**.** *For any real number $r > \omega(\sqrt{\log n})$, we have*

$$\Pr_{x \leftarrow D_{\mathbb{Z}^n,r}} \left[||x|| > r\sqrt{n} \right] \leq 2^{-n+1}$$

Using Lemma 2.6.3 together with the fact that for all $\mathbf{x} \in \mathbb{R}^n$, $\|\mathbf{x}\| \geq \|\mathbf{x}\|_\infty$ yields the following bound.

Lemma 2.6.4. *Let $n = \omega(\log \kappa)$. For any real number $r > \omega(\sqrt{\log n})$, we have*

$$\Pr_{\mathbf{x} \leftarrow D_{\mathbb{Z}^n,r}} \left[\|\mathbf{x}\|_\infty > r\sqrt{n} \right] \leq 2^{-n+1} = \mathsf{negl}(\kappa)$$

Define the *truncated Discrete Gaussian distribution* with standard deviation r and centered at $\mathbf{c}$, denoted by $\overline{D}_{\mathbb{Z}^n,r,\mathbf{c}}$, to be one that samples a polynomial according to the discrete Gaussian $D_{\mathbb{Z}^n,r,\mathbf{c}}$ and repeats the sampling if the polynomial is not $(r\sqrt{n})$-bounded. As long as $n = \omega(\log(\kappa))$, Lemma 2.6.4 implies that this distribution is statistically close to the discrete Gaussian : $\overline{D}_{\mathbb{Z}^n,r,\mathbf{c}} \approx_s D_{\mathbb{Z}^n,r,\mathbf{c}}$.

The second fact says that the statistical distance between a discrete Gaussian with standard deviation r and centered at 0, and one centered at $\mathbf{c} \in \mathbb{Z}^n$ is at most $\|\mathbf{c}\|/r$. In particular, if r is super-polynomially larger than $\|\mathbf{c}\|$ then the two distributions are statistically close.

Lemma 2.6.5. *Let $n \in \mathbb{N}$. For any real number $r > \omega(\sqrt{\log n})$, and any $\mathbf{c} \in \mathbb{Z}^n$, the statistical distance between the distributions $D_{\mathbb{Z}^n,r}$ and $D_{\mathbb{Z}^n,r,\mathbf{c}}$ is at most $||\mathbf{c}||/r$.*

Corollary 2.6.6. *Let $\mathbf{c} \in \mathbb{Z}^n$. For any real number $r \geq 2^{\omega(\log \kappa)} \|\mathbf{c}\|$, the distributions $D_{\mathbb{Z}^n,r}$ and $D_{\mathbb{Z}^n,r,\mathbf{c}}$ are statistically close.*

2.6.2 The Ring LWE Assumption

We now describe the Ring Learning With Errors (RLWE) assumption of Lyubaskevsky, Peikert, and Regev [LPR10]. The RLWE assumption is analogous to the standard Learning With Errors (LWE) assumption, first defined by Regev [Reg05, Reg09] (generalizing the learning parity with noise assumption of Blum et al. [BFKL93]).

The $\mathsf{RLWE}_{\phi,q,\chi}$ assumption is that for a random ring element $s \leftarrow R_q$, given any polynomial number of samples of the form $(a_i, b_i = a_i \cdot s + e_i) \in R_q^2$, where a_i is uniformly random in R_q and e_i is drawn from the error distribution χ, the b_i's are computationally indistinguishable from uniform in R_q. We use the *Hermite normal form* of the assumption, as in [BV11b], where the secret s is sampled from the noise distribution χ rather than being uniform in R_q. This presentation is more useful for the purposes of this work and is equivalent to the original up to obtaining one additional sample [ACPS09, LPR10].

Definition 2.6.4. (The RLWE Assumption - Hermite Normal Form [LPR10]) *For all $\kappa \in \mathbb{N}$, let $\phi(x) = \phi_\kappa(x) \in \mathbb{Z}[x]$ be a polynomial of degree $n = n(\kappa)$, let $q = q(\kappa) \in \mathbb{Z}$ be an odd prime integer, let the ring $R \stackrel{\mathsf{def}}{=} \mathbb{Z}[x]/\langle \phi(x) \rangle$ and $R_q \stackrel{\mathsf{def}}{=} R/qR$, and let χ denote a distribution over the ring R.*

The Decisional Ring LWE assumption $\mathsf{RLWE}_{\phi,q,\chi}$ *states that for any $\ell = \mathsf{poly}(\kappa)$ it holds that*

$$\{(a_i, a_i \cdot s + e_i)\}_{i \in [\ell]} \stackrel{c}{\approx} \{(a_i, u_i)\}_{i \in [\ell]} \ ,$$

where s is sampled from the noise distribution χ, a_i are uniform in R_q, the "error polynomials" e_i are sampled from the error distribution χ, and finally, the ring elements u_i are uniformly random over R_q.

We now present a couple of facts about the RLWE assumption. The first says that the assumption also holds if the error is multiplied by 2 in every sample. This follows immediately from the fact that q is an odd prime and therefore relatively prime with 2.

Fact 2.6.7. *The $\mathsf{RLWE}_{\phi,q,\chi}$ assumption implies that for any $\ell = \mathsf{poly}(\kappa)$,*

$$\{(a_i, a_i \cdot s + 2 \cdot e_i)\}_{i \in [\ell]} \stackrel{c}{\approx} \{(a_i, u_i)\}_{i \in [\ell]} \ .$$

where a_i, u_i are uniformly random in R_q and s, e_i are drawn from the error distribution χ.

The second fact says that the assumption also holds if the distinguisher is additionally given samples with the same parameter $\mathbf{a}_j$ and different secret key s_i. This follows from a hybrid argument that slowly changes the samples, one secret at a time, from RLWE to uniform.

Fact 2.6.8. *The $\mathsf{RLWE}_{\phi,q,\chi}$ assumption implies that for any $\ell = \mathsf{poly}(\kappa), \ell' = \mathsf{poly}(\kappa)$,*

$$\{(a_j, a_j \cdot s_i + e_{i,j})\}_{i \in [\ell], j \in [\ell']} \stackrel{c}{\approx} \{(a_j, u_{i,j})\}_{i \in [\ell], j \in [\ell']} \ .$$

where $a_j, u_{i,j}$ are uniformly random in R_q and $s_i, e_{i,j}$ are drawn from the error distribution χ.

2.6.3 Choice of Parameters

As already stated above, we will rely of the following specific choices of the polynomial $\phi(x)$ and the error distribution χ. For security parameter κ and a dimension parameter $n = n(\kappa)$ which is a power of two:

- We set $\phi(x) \stackrel{\mathsf{def}}{=} x^n + 1$ where n is a power of two.
- The error distribution χ is the truncated discrete Gaussian distribution $\overline{D}_{\mathbb{Z}^n,r}$ with standard deviation $r > 0$. A sample from this distribution is a $(r\sqrt{n})$-bounded polynomial $e \in R$.

2.6.4 The Worst-case to Average-case Connection

We state a worst-case to average-case reduction from the shortest vector problem on ideal lattices to the RLWE problem for our setting of parameters. The reduction stated below is a special case of the results of [LPR10].

Theorem 2.6.9 ([LPR10])**.** *Let $\phi(x) = x^n + 1$ where n is a power of two. Let $r \geq \omega(\sqrt{\log n})$ be a real number, and let $q \equiv 1 \pmod{2n}$ be a prime integer. Let $R \stackrel{\mathsf{def}}{=} \mathbb{Z}[x]/\langle\phi(x)\rangle$. Then there is a randomized reduction from $2^{\omega(\log n)} \cdot (q/r)$-approximate R-SVP to $\mathsf{RLWE}_{\phi,q,\chi}$ where $\chi = D_{\mathbb{Z}^n,r}$ is the discrete Gaussian distribution.*

Solving approximate R-SVP to within a sub-exponential factor is believed to be hard. Thus, if $q/r = 2^{o(n)}$ then the $\mathsf{RLWE}_{\phi,q,\chi}$ assumption is believed to be hard.

2.7 FHE from RLWE

We describe a leveled homomorphic encryption scheme based on the RLWE assumption. Our starting point is the scheme of Brakerski and Vaikuntanathan [BV11b], which is based on (ring-based) Regev encryption [LPR10, LPR13]. However, we use the relinearization/key-switching technique of Brakerski et al. [BV11a, BGV12] to ensure that the ciphertext size remains constant, and the techniques of Brakerski [Bra12] for noise management. This is in contrast with the "squashing" and "bootstrapping" blueprint of Gentry [Gen09b] used in the work of Brakerski and Vaikuntanathan [BV11b]. We remark that this modified scheme has already been described by Fan and Vercauteren [FV12].

2.7.1 Regev Encryption in Rings

For security parameter κ, the scheme is parameterized by a prime number $q = q(\kappa)$, a degree $n = n(\kappa)$ polynomial $\phi(x) \in \mathbb{Z}[x]$, and an error distribution $\chi = \chi(\kappa)$ over the ring $R \stackrel{\mathsf{def}}{=} \mathbb{Z}[x]/\langle\phi(x)\rangle$. The parameters n, ϕ, q and χ are public and we assume that given κ, there are polynomial-time algorithms that output ϕ and q, and sample from the error distribution χ. The message space is $\mathcal{M} = \{0, 1\}$, and all operations are carried out in the ring R (i.e. modulo $\phi(x)$).

- $\mathsf{Setup}(1^\kappa)$: Sample a ring element $a \leftarrow R_q$, and set $\mathsf{params} := a$.
- $\mathsf{Keygen}(\mathsf{params})$: Sample a ring element $s \leftarrow \chi$ and a ring element $x \leftarrow \chi$, and set

$$\mathsf{sk} := \mathbf{s} = (1, s) \quad \text{and} \quad \mathsf{pk} := p = [as + x]_q \in R_q$$

- $\mathsf{Enc}(\mathsf{pk}, m)$: To encrypt a bit $m \in \{0,1\}$ with public key $\mathsf{pk} = p$, sample $r, e_1, e_2 \leftarrow \chi$, compute

$$v := \left[pr + e_1 + \left\lfloor \frac{q}{2} \right\rfloor m\right]_q \in R_q \qquad \text{and} \qquad w := [-ar + e_2]_q \in R_q$$

  and output the ciphertext $\mathbf{c} := (v, w) \in R_q^2$.

- $\mathsf{Dec}(\mathsf{sk}, \mathbf{c})$: To decrypt ciphertext $\mathbf{c}$ using secret key $\mathsf{sk} = \mathbf{s}$, output

$$m = \left[\left\lfloor \frac{2}{q} \cdot [\langle \mathbf{c}, \mathbf{s} \rangle]_q \right\rceil\right]_2$$

Semantic security of the scheme follows from the fact that p is a RLWE sample and therefore pseudorandom, and the fact that if p is random then $pr + e_1$ and $-ar + e_2$ are RLWE samples and therefore pseudorandom as well.

2.7.2 Homomorphism

In order to perform homomorphic operations, we will use use two subroutines that, given two vectors $\mathbf{c}$ and $\mathbf{s}$, "expand" these vectors to get longer (higher-dimensional) vectors $\mathbf{c}'$ and $\mathbf{s}'$ such that $[\langle \mathbf{c}', \mathbf{s}' \rangle]_q = [\langle \mathbf{c}, \mathbf{s} \rangle]_q$. We describe these subroutines first. For any $\ell \in \mathbb{N}$ and any $\mathbf{x} \in R_q^\ell$;

- $\mathsf{Bit}(\mathbf{x})$ decomposes $\mathbf{x}$ into its bit representation. Namely,
  $\mathsf{Bit}(\mathbf{x}) = (\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{\lfloor \log q \rfloor}) \in R_2^{\ell \cdot \lceil \log q \rceil}$ such that $\mathbf{x} = \sum_{j=0}^{\lfloor \log q \rfloor} 2^j \cdot \mathbf{x}_j$.

- $\mathsf{Pow}(\mathbf{x}) = \left[(\mathbf{x}, 2 \cdot \mathbf{x}, \ldots, 2^{\lfloor \log q \rfloor} \cdot \mathbf{x})\right]_q \in R_q^{\ell \cdot \lceil \log q \rceil}$.

Observe that for all $\mathbf{c}, \mathbf{s} \in R_q^\ell$:

$$\begin{aligned}
[\langle \mathsf{Bit}(\mathbf{c}), \mathsf{Pow}(\mathbf{s}) \rangle]_q &= \left[\sum_{j=0}^{\lfloor \log q \rfloor} \langle \mathbf{c}_j, 2^j \cdot \mathbf{s} \rangle\right]_q = \left[\sum_{j=0}^{\lfloor \log q \rfloor} \langle 2^j \cdot \mathbf{c}_j, \mathbf{s} \rangle\right]_q \\
&= \left[\left\langle \sum_{j=0}^{\lfloor \log q \rfloor} 2^j \cdot \mathbf{c}_j, \mathbf{s} \right\rangle\right]_q \\
&= [\langle \mathbf{c}, \mathbf{s} \rangle]_q
\end{aligned}$$

Now consider a ciphertext $\mathbf{c}$ encrypting a message m, and note that $\langle \mathbf{c}, \mathbf{s} \rangle = \frac{q}{2} \cdot m + e + qI$ for some $I \in R$ and $e \in \mathbb{Q}[x]/\langle x^n + 1 \rangle$ with $\|e\|_\infty \leq E$ for some bound $E < q/4$. As long as a ciphertext has this form, decryption will correctly yield the underlying message m. We can view homomorphic operations in this light.

Addition. For ciphertexts $\mathbf{c}_1 = (v_1, w_1)$ and $\mathbf{c}_2 = (v_2, w_2)$ such that $\langle \mathbf{c}_1, \mathbf{s} \rangle = \frac{q}{2} \cdot m_1 + e_1 + qI_1$ and $\langle \mathbf{c}_2, \mathbf{s} \rangle = \frac{q}{2} \cdot m_2 + e_2 + qI_2$, we have:

$$\langle \mathbf{c}_1 + \mathbf{c}_2, \mathbf{s} \rangle = \langle \mathbf{c}_1, \mathbf{s} \rangle + \langle \mathbf{c_2}, \mathbf{s} \rangle = \frac{q}{2} \cdot (m_1 + m_2) + (e_1 + e_2) + q(I_1 + I_2)$$

Therefore, it is natural to define $\mathbf{c}_{\mathsf{add}} = \mathbf{c}_1 + \mathbf{c}_2 = (v_1 + v_2, w_1 + w_2)$.

Multiplication. Multiplication is more tricky. For ciphertexts $\mathbf{c}_1 = (v_1, w_1)$ and $\mathbf{c}_2 = (v_2, w_2)$ such that $\langle \mathbf{c}_1, \mathbf{s} \rangle = \frac{q}{2} \cdot m_1 + e_1 + qI_1$ and $\langle \mathbf{c}_2, \mathbf{s} \rangle = \frac{q}{2} \cdot m_2 + e_2 + qI_2$, we have:

$$\begin{aligned}\left\langle \frac{2}{q} \cdot \mathbf{c}_1 \otimes \mathbf{c}_2, \mathbf{s} \otimes \mathbf{s} \right\rangle = \frac{2}{q} \cdot \langle \mathbf{c}_1, \mathbf{s} \rangle \langle \mathbf{c}_2, \mathbf{s} \rangle &= \frac{2}{q} \cdot \left(\frac{q}{2} \cdot m_1 + e_1 + qI_1 \right) \left(\frac{q}{2} \cdot m_2 + e_2 + qI_2 \right) \\ &= \frac{q}{2} \cdot m_1 m_2 + e_{\mathsf{mult}} + q(m_2 I_1 + m_2 I_1 + 2qI_1 I_2)\end{aligned}$$

where $e_{\mathsf{mult}} = (m_1 e_2 + e_1 m_2 + 2e_1 I_2 + 2e_2 I_1 + \frac{2}{q} \cdot e_1 e_2)$.

It is therefore natural to define $\mathbf{c}_{\mathsf{mult}} = \left\lfloor \frac{2}{q} \cdot \mathbf{c}_1 \otimes \mathbf{c}_2 \right\rceil$ as a ciphertext encrypting $m_1 m_2$ under secret key $\mathbf{s} \otimes \mathbf{s}$. Unfortunately, defining $\mathbf{c}_{\mathsf{mult}}$ in this way forces the ciphertext and secret-key sizes to grow exponentially with the number of multiplications. Alternatively, notice that

$$\left\langle \frac{2}{q} \cdot \mathbf{c}_1 \otimes \mathbf{c}_2, \mathbf{s} \otimes \mathbf{s} \right\rangle \approx \beta_0 + \beta_1 s + \beta_2 s^2 = \left\langle (\beta_0, \beta_1, \beta_2), (1, s, s^2) \right\rangle$$

for

$$\beta_0 = \left\lfloor \frac{2}{q} \cdot v_1 v_2 \right\rceil \quad , \quad \beta_1 = \left\lfloor \frac{2}{q} \cdot (v_1 w_2 + w_1 v_2) \right\rceil \quad , \quad \beta_2 = \left\lfloor \frac{2}{q} \cdot w_1 w_2 \right\rceil$$

If defined this way, the ciphertext and secret-key sizes grow only linearly with the number of multiplications. Indeed, this is the approach taken in the work of Brakerski and Vaikutanathan [BV11b], albeit with a different noise-reduction technique.

In our setting of multiparty computation, even this linear growth in the secret-key size would make our protocols very complex, so we opt for another strategy. Instead of letting the ciphertext grow with each multiplication, we use the relinearization/key-switching technique of Brakerski et al. [BV11a, BGV12] to reduce the size of the resulting ciphertext after each multiplication. The result is a ciphertext $\mathbf{c}_{\mathsf{mult}}$ that encrypts the product mm' under a *new key* $\mathbf{t} = (1, t)$. This step requires the aid of a public evaluation key, as follows.

Evaluation Key. Sample $t \leftarrow \chi$ and set $\mathbf{t} = (1, t)$. Sample $\boldsymbol{\alpha}, \widetilde{\boldsymbol{\alpha}} \leftarrow R_q^{\lceil \log q \rceil}$ and $\boldsymbol{\varepsilon}, \widetilde{\boldsymbol{\varepsilon}} \leftarrow \chi^{\lceil \log q \rceil}$, and compute

$$\begin{aligned}\boldsymbol{\gamma} &:= \left[\boldsymbol{\alpha} t + \boldsymbol{\varepsilon} + \mathsf{Pow}(s) \right]_q \in R_q^{\lceil \log q \rceil} \quad , \quad \boldsymbol{\zeta} := -\boldsymbol{\alpha} \\ \widetilde{\gamma} &:= \left[\widetilde{\boldsymbol{\alpha}} t + \widetilde{\boldsymbol{\varepsilon}} + \mathsf{Pow}\left(s^2\right) \right]_q \in R_q^{\lceil \log q \rceil} \quad , \quad \widetilde{\boldsymbol{\zeta}} := -\widetilde{\boldsymbol{\alpha}}\end{aligned}$$

The evaluation key is defined to be $\mathsf{ek} := \left(\boldsymbol{\gamma}, \boldsymbol{\zeta}, \widetilde{\boldsymbol{\gamma}}, \widetilde{\boldsymbol{\zeta}} \right)$. The reader may notice that $(\boldsymbol{\gamma}, \boldsymbol{\zeta})$ and $\left(\widetilde{\boldsymbol{\gamma}}, \widetilde{\boldsymbol{\zeta}} \right)$ are private-key *pseudo-encryptions* of $\mathsf{Pow}(s)$ and $\mathsf{Pow}\left(s^2\right)$, respectively, under secret key $\mathbf{t}$. We say they are pseudo-encryptions because even though they were created as normal ciphertexts, they cannot be decrypted in the usual way since the elements of $\mathsf{Pow}(s)$ and $\mathsf{Pow}\left(s^2\right)$ are not in the message space $\{0, 1\}$.

Relinearization/Key-Switching. For ciphertexts $\mathbf{c}_1 = (v_1, w_1)$ and $\mathbf{c}_2 = (v_2, w_2)$ such that $\langle \mathbf{c}_1, \mathbf{s} \rangle = \frac{q}{2} \cdot m_1 + e_1 + qI_1$ and $\langle \mathbf{c}_2, \mathbf{s} \rangle = \frac{q}{2} \cdot m_2 + e_2 + qI_2$, define $\beta_0, \beta_1, \beta_2 \in R$ as before:

$$\beta_0 = \left\lfloor \frac{2}{q} \cdot v_1 v_2 \right\rceil \quad , \quad \beta_1 = \left\lfloor \frac{2}{q} \cdot (v_1 w_2 + w_1 v_2) \right\rceil \quad , \quad \beta_2 = \left\lfloor \frac{2}{q} \cdot w_1 w_2 \right\rceil$$

so that $\beta_0 + \beta_1 s + \beta_2 s^2 \approx \left\langle \frac{2}{q} \cdot \mathbf{c}_1 \otimes \mathbf{c}_2, \mathbf{s} \otimes \mathbf{s} \right\rangle = \frac{2}{q} \langle \mathbf{c}_1, \mathbf{s} \rangle \langle \mathbf{c}_2, \mathbf{s} \rangle$. Output $\mathbf{c}_{\mathsf{mult}} = (v_{\mathsf{mult}}, w_{\mathsf{mult}})$ where

$$\begin{aligned} v_{\mathsf{mult}} &= \left[\beta_0 + \left\langle \mathsf{Bit}\left(\beta_1, \beta_2\right), (\boldsymbol{\gamma}, \widetilde{\boldsymbol{\gamma}}) \right\rangle\right]_q \\ w_{\mathsf{mult}} &= \left[- \left\langle \mathsf{Bit}\left(\beta_1, \beta_2\right), \left(\boldsymbol{\zeta}, \widetilde{\boldsymbol{\zeta}}\right) \right\rangle \right]_q \end{aligned}$$

The ciphertext $\mathbf{c}_{\mathsf{mult}}$ is a valid encryption of $m_1 m_2$ under secret key $\mathbf{t}$. To see why this is the case, simply notice that:

$$\begin{aligned} \left[\langle \mathbf{c}_{\mathsf{mult}}, \mathbf{t} \rangle\right]_q &= \left[\beta_0 + \left(\langle \mathsf{Bit}\left(\beta_1\right), \boldsymbol{\gamma} \rangle - \langle \mathsf{Bit}\left(\beta_1\right), \boldsymbol{\zeta} \rangle \cdot t\right)\right. \\ &\qquad \left. + \left(\langle \mathsf{Bit}\left(\beta_2\right), \widetilde{\boldsymbol{\gamma}} \rangle - \left\langle \mathsf{Bit}\left(\beta_2\right), \widetilde{\boldsymbol{\zeta}} \right\rangle \cdot t\right)\right]_q \\ &= \left[\beta_0 + \left(\langle \mathsf{Bit}\left(\beta_1\right), \mathsf{Pow}\left(s\right) \rangle + x\right) + \left(\left\langle \mathsf{Bit}\left(\beta_2\right), \mathsf{Pow}\left(s^2\right) \right\rangle + \widetilde{x}\right)\right]_q \\ &= \left[\beta_0 + \beta_1 s + \beta_2 s^2 + (x + \widetilde{x})\right]_q \end{aligned}$$

where $x \stackrel{\mathsf{def}}{=} \langle \mathsf{Bit}\left(\beta_1\right), \boldsymbol{\varepsilon} \rangle$ and $\widetilde{x} \stackrel{\mathsf{def}}{=} \langle \mathsf{Bit}\left(\beta_2\right), \widetilde{\boldsymbol{\varepsilon}} \rangle$ are "small" error terms. This means that for some $I \in R$:

$$\begin{aligned} \langle \mathbf{c}_{\mathsf{mult}}, \mathbf{t} \rangle &= \beta_0 + \beta_1 s + \beta_2 s^2 + (x + \widetilde{x}) + qI \\ &\approx \frac{2}{q} \cdot \langle \mathbf{c}_1, \mathbf{s} \rangle \langle \mathbf{c}_2, \mathbf{s} \rangle + (x + \widetilde{x}) + qI \\ &= \frac{q}{2} \cdot m_1 m_2 + \widetilde{e} + q\widetilde{I} \end{aligned}$$

where $\widetilde{e} \in \mathbb{Q}[x]/\langle x^n + 1 \rangle$ is "small" and $\widetilde{I} \in R$.

Remark 2.7.1. *From the discussion above and the fact that the coefficients of* $\mathsf{Bit}\left(\cdot\right)$ *are bounded by* 1*, it is easy to see that relinearization increases the error of a ciphertext by an additive factor of*

$$\|x + \widetilde{x}\|_\infty \leq \|\langle \mathsf{Bit}\left(\beta_1\right), \boldsymbol{\varepsilon} \rangle\|_\infty + \|\langle \mathsf{Bit}\left(\beta_2\right), \widetilde{\boldsymbol{\varepsilon}} \rangle\|_\infty \leq n \left\lceil \log q \right\rceil \left(\|\boldsymbol{\varepsilon}\|_\infty + \|\widetilde{\boldsymbol{\varepsilon}}\|_\infty\right)$$

2.7.3 Scale-Invariant FHE from RLWE

For security parameter κ, the scheme is parameterized by a prime number $q = q(\kappa)$, a degree $n = n(\kappa)$ polynomial $\phi(x) \in \mathbb{Z}[x]$, and an error distribution $\chi = \chi(\kappa)$ over the ring $R \stackrel{\mathsf{def}}{=} \mathbb{Z}[x]/\langle \phi(x) \rangle$. The scheme is also parametrized by an integer value D that corresponds to the maximum circuit depth that the scheme is able to homomorphically evaluate. The parameters n, ϕ, q, χ, D are public and we assume that given κ, there are polynomial-time algorithms that output ϕ and q, and sample from the error distribution χ. The message space is $\mathcal{M} = \{0, 1\}$, and all operations are carried out in the ring R (i.e. modulo $\phi(x)$).

- $\mathsf{Setup}(1^\kappa)$: Sample ring element $a^{(0)} \leftarrow R_q$, and for $d \in \{1, \ldots, D\}$, sample $a^{(d)} \leftarrow R_q$ and $\boldsymbol{\alpha}^{(d)}, \widetilde{\boldsymbol{\alpha}}^{(d)} \leftarrow R_q^{\lceil \log q \rceil}$. Set

$$\mathsf{params} := \left(a^{(0)}, \left\{a^{(d)}, \boldsymbol{\alpha}^{(d)}, \widetilde{\boldsymbol{\alpha}}^{(d)}\right\}_{d \in [D]}\right)$$

- $\mathsf{Keygen}(\mathsf{params})$: Sample ring elements $s^{(0)}, x^{(0)} \leftarrow \chi$. For $d \in [D]$, sample a ring elements $s^{(d)}, x^{(d)} \leftarrow \chi$, and vectors $\boldsymbol{\varepsilon}^{(d)}, \widetilde{\boldsymbol{\varepsilon}}^{(d)} \leftarrow \chi^{\lceil \log q \rceil}$, and compute

$$\boldsymbol{\gamma}^{(d)} := \left[\boldsymbol{\alpha}^{(d)} \cdot s^{(d)} + \boldsymbol{\varepsilon}^{(d)} + \mathsf{Pow}\left(s^{(d-1)}\right)\right]_q \in R_q^{\lceil \log q \rceil} \quad , \quad \boldsymbol{\zeta}^{(d)} := -\boldsymbol{\alpha}^{(d)}$$

$$\widetilde{\boldsymbol{\gamma}}^{(d)} := \left[\widetilde{\boldsymbol{\alpha}}^{(d)} \cdot s^{(d)} + \widetilde{\boldsymbol{\varepsilon}}^{(d)} + \mathsf{Pow}\left(\left(s^{(d-1)}\right)^2\right)\right]_q \in R_q^{\lceil \log q \rceil} \quad , \quad \widetilde{\boldsymbol{\zeta}}^{(d)} := -\widetilde{\boldsymbol{\alpha}}^{(d)}$$

  For all $d \in \{0, \ldots, D\}$, define $\mathbf{s}^{(d)} = \left(1, s^{(d)}\right)$ and $p^{(d)} = \left[a^{(d)} s^{(d)} + x^{(d)}\right]_q$, and set

$$\mathsf{sk} := \left(\mathbf{s}^{(0)}, \ldots, \mathbf{s}^{(D)}\right) \ , \ \mathsf{pk} := \left(p^{(0)}, \ldots, p^{(D)}\right) \ , \ \mathsf{ek} := \left\{\boldsymbol{\gamma}^{(d)}, \boldsymbol{\zeta}^{(d)}, \widetilde{\boldsymbol{\gamma}}^{(d)}, \widetilde{\boldsymbol{\zeta}}^{(d)}\right\}_{d \in [D]}$$

- $\mathsf{Enc}(\mathsf{pk}, m)$: To encrypt a bit $m \in \{0,1\}$ with public key $\mathsf{pk} = (p^{(0)}, \ldots, p^{(D)})$, sample $r, e_1, e_2 \leftarrow \chi$, and compute

$$v := \left[pr + e_1 + \left\lfloor \frac{q}{2} \right\rfloor m\right]_q \in R_q \qquad \text{and} \qquad w := \left[-a^{(0)} r + e_2\right]_q \in R_q$$

  and output the ciphertext $\mathbf{c} := (v, w) \in R_q^2$.

- $\mathsf{Dec}(\mathsf{sk}, \mathbf{c})$: To decrypt ciphertext $\mathbf{c}$ using secret key $\mathsf{sk} = \left(\mathbf{s}^{(0)}, \ldots, \mathbf{s}^{(D)}\right)$, output

$$m = \left[\left\lfloor \frac{2}{q} \cdot \left[\left\langle \mathbf{c}, \mathbf{s}^{(D)} \right\rangle\right]_q \right\rceil\right]_2$$

- $\mathsf{Eval}(C, (\mathbf{c}_1, \ldots, \mathbf{c}_\ell))$: We show how to evaluate an ℓ-variate boolean circuit $C : \{0,1\}^\ell \to \{0,1\}$ of depth D. To this end, we show how to homomorphically add and multiply two elements in $\{0,1\}$.

  - Given two ciphertexts $\mathbf{c}_1 = (v_1, w_1)$ and $\mathbf{c}_2 = (v_2, w_2)$ *under the same secret key* $s^{(d-1)}$, output the ciphertext $\mathbf{c}_{\mathsf{add}} = \mathbf{c}_1 + \mathbf{c}_2 = (v_1 + v_2, w_1 + w_2) \in R_q^2$, as an encryption of the *sum* of the underlying messages.
  - Given two ciphertexts $\mathbf{c}_1 = (v_1, w_1)$ and $\mathbf{c}_2 = (v_2, w_2)$ *under the same secret key* $s^{(d-1)}$, output the ciphertext $\mathbf{c}_{\mathsf{mult}} = (v_{\mathsf{mult}}, w_{\mathsf{mult}}) \in R_q^2$, as an encryption of the *product* of the underlying messages, where:

$$v_{\mathsf{mult}} = \left[\beta_0 + \left\langle \mathsf{Bit}\left(\beta_1, \beta_2\right), \left(\boldsymbol{\gamma}^{(d)}, \widetilde{\boldsymbol{\gamma}}^{(d)}\right)\right\rangle\right]_q$$
$$w_{\mathsf{mult}} = \left[-\left\langle \mathsf{Bit}\left(\beta_1, \beta_2\right), \left(\boldsymbol{\zeta}^{(d)}, \widetilde{\boldsymbol{\zeta}}^{(d)}\right)\right\rangle\right]_q$$

  and $\beta_0 = \left\lfloor \frac{2}{q} \cdot v_1 v_2 \right\rceil$, $\beta_1 = -\left\lfloor \frac{2}{q} \cdot (v_1 w_2 + w_1 v_2) \right\rceil$, $\beta_2 = \left\lfloor \frac{2}{q} \cdot w_1 w_2 \right\rceil$.

Following the analysis of Brakerski [Bra12] yields the following theorem. We choose to use the bounds on the size of the secret key and the error in the evaluation key, because these will be important in the analysis of our constructions.

Theorem 2.7.1. *The error in a ciphertext that is the outcome of a depth-D circuit evaluation is bounded by*

$$O\left((n^3 B_{\mathrm{SK}})^D \left(nB_{\mathrm{ENC}}^2 + nB_{\mathrm{ENC}}B_{\mathrm{SK}} + n^2 B_{\mathrm{SK}}^2 + n\lceil \log q \rceil \cdot B_{\mathrm{EK}}\right)\right)$$

where we assume that $s^{(d)}$ is B_{SK}-bounded for all $d \in \{0,\ldots,D\}$, χ is a B_{ENC}-bounded distribution, and $\boldsymbol{\varepsilon}^{(d)}, \widetilde{\boldsymbol{\varepsilon}}^{(d)}$ are B_{EK}-bounded for all $d \in [D]$.

Corollary 2.7.2. *Let χ be a B-bounded distribution. Then the encryption scheme described above can evaluate depth-D circuits as long as $q = \Omega\left((n^3 B)^D \cdot \left(n^2 B^2 + nB\lceil \log q \rceil\right)\right)$. The security of the scheme is based on the $\mathsf{RLWE}_{\phi,q,\chi}$ assumption, which is believed to be hard as long as $q/B = 2^{o(n)}$.*

2.8 NTRU Encryption

We describe the NTRU encryption scheme of Hofftein et al. [HPS98], with the modifications proposed by Stehlé and Steinfeld [SS11b]. For security parameter κ, the scheme is parameterized by a prime number $q = q(\kappa)$, a degree $n = n(\kappa)$ polynomial $\phi(x) \in \mathbb{Z}[x]$, and an error distribution $\chi = \chi(\kappa)$ over the ring $R \stackrel{\mathsf{def}}{=} \mathbb{Z}[x]/\langle \phi(x) \rangle$. The parameters n, ϕ, q, χ are public and we assume that given κ, there are polynomial-time algorithms that output ϕ and q, and sample from the error distribution χ. The message space is $\mathcal{M} = \{0,1\}$, and all operations are carried out in the ring R (i.e. modulo $\phi(x)$).

- $\mathsf{Keygen}(1^\kappa)$: Sample polynomials $f', g \leftarrow \chi$ and set $f \stackrel{\mathsf{def}}{=} 2f' + 1$ so that $f \equiv 1 \pmod 2$. If f is not invertible in R_q, resample f'; otherwise, let f^{-1} be the inverse of f in R_q. Set

$$\mathsf{pk} \stackrel{\mathsf{def}}{=} h := \left[2gf^{-1}\right]_q \in R_q \quad , \quad \mathsf{sk} \stackrel{\mathsf{def}}{=} f \in R$$

- $\mathsf{Enc}(\mathsf{pk}, m)$: To encrypt a bit $m \in \{0,1\}$ with public key $\mathsf{pk} = h$, sample polynomials $s, e \leftarrow \chi$ and output the ciphertext

$$c \stackrel{\mathsf{def}}{=} [hs + 2e + m]_q \in R_q$$

- $\mathsf{Dec}(\mathsf{sk}, c)$: To decrypt a ciphertext $c \in R_q$ with secret key $\mathsf{sk} = f$, let $\mu \stackrel{\mathsf{def}}{=} [fc]_q$ and output $m \stackrel{\mathsf{def}}{=} \mu \pmod 2$.

It is easily seen that this scheme is correct as long as there is no reduction modulo q. To decrypt a ciphertext c, we compute:

$$[fc]_q = [fhs + 2fe + fm]_q = [2gs + 2fe + fm]_q$$

If there is no reduction modulo q then

$$[fc]_q \pmod 2 = 2gs + 2fe + fm \pmod 2 = fm \pmod 2 = m$$

Furthermore, our choice of parameter $\phi(x) = x^n + 1$ ensures there is no reduction modulo q. Notice that since the coefficients of g, s, e are all bounded by B, and the coefficients of f are bounded by $2B + 1$. By Corollary 2.6.2, we know that the coefficients of $[fc]_q$ are bounded by $2nB^2(2nB + 1)(2B + 1)$. As long as we set q to be large enough so that $q/2$ is larger than this quantity, a fresh ciphertext generated by Enc is guaranteed to decrypt correctly. From here on, we refer to $\mu = [fc]_q \in R_q$ as the "*error* in ciphertext c".

2.8.1 Security

The security of the (modified) NTRU encryption scheme can be based on two assumptions – the RLWE assumption described in Section 2.6, as well as an assumption that we call the (Decisional) Small Polynomial Ratio (DSPR) Assumption.

Definition 2.8.1. (Decisional Small Polynomial Ratio Assumption) *Let $\phi(x) \in \mathbb{Z}[x]$ be a polynomial of degree n, let $q \in \mathbb{Z}$ be a prime integer, and let χ denote a distribution over the ring $R \stackrel{\mathsf{def}}{=} \mathbb{Z}[x]/\langle\phi(x)\rangle$. The* (decisional) small polynomial ratio assumption $\mathsf{DSPR}_{\phi,q,\chi}$ *says that it is hard to distinguish the following two distributions:*

- *a polynomial $h \stackrel{\mathsf{def}}{=} \left[2gf^{-1}\right]_q$, where f' and g are sampled from the distribution χ (conditioned on $f \stackrel{\mathsf{def}}{=} 2f+1$ being invertible over R_q) and f^{-1} is the inverse of f in R_q.*
- *a polynomial u sampled uniformly at random over R_q.*

The security proof uses a hybrid argument, in two steps.

1. The hardness of $\mathsf{DSPR}_{\phi,q,\chi}$ allows to change the public key $h = \left[2gf^{-1}\right]_q$ to a uniformly sampled h.
2. Once this is done, we can use $\mathsf{RLWE}_{\phi,q,\chi}$ to change the challenge ciphertext $c^* = [hs + 2e + m]_q$ to $c^* = [u + m]_q$, where u is uniformly sampled from R_q.

   In this final hybrid, the advantage of the adversary is exactly $1/2$ since c^* is uniform over R_q, independent of the message m.

Stehlé and Steinfeld [SS11b] showed that the $\mathsf{DSPR}_{\phi,q,\chi}$ assumption is unconditionally true even for unbounded adversaries (namely, the two distributions above are statistically close) if n is a power of two, $\phi(x) = x^n + 1$, and χ is the discrete Gaussian $D_{\mathbb{Z}^n,r}$ for $r > \sqrt{q} \cdot \mathsf{poly}(n)$. Thus, with this setting of parameters, semantic security of the modified NTRU scheme can be based on the $\mathsf{RLWE}_{\phi,q,\chi}$ assumption alone.

Chapter 3
New Notions in MPC

In this chapter, we introduce two new notions of secure multiparty computation (MPC) [GMW87, BGW88, CCD88], *cloud-assisted MPC* and *on-the-fly MPC*, specifically tailored to the *delegation* of multiparty computation to a powerful but untrusted cloud server.

3.1 Cloud-Assisted MPC

We consider a variant of MPC, where N parties wish to securely compute a joint function of their inputs, but want to do so in a way such that the amount of data they interchange, as well as their computation time, is independent of the complexity of the function. In order to do this, we rely on the computation power of a new party called the *server* or *cloud*, who will perform the computation of the function f. The computation should remain secure even when the server and any colluding subset of parties are corrupted. We formalize this idea below.

3.1.1 Definition

For an N-input function f, we define a *cloud-assisted multiparty protocol* Π for f to be an MPC protocol between N interactive Turing Machines $P_1, \ldots, P_N$, called *parties* or *clients*, and a *server* S, also an interactive Turing Machine. We require that for all $\vec{x} = (x_1, \ldots, x_N)$, the output of Π in an execution where P_i is given x_i as input (and the server S does not receive an input), is $y \stackrel{\mathsf{def}}{=} f(\vec{x})$.

Furthermore, we require the communication complexity of the protocol, as well as the computation time of the clients $P_1, \ldots, P_N$, to be *(essentially) independent* of the complexity of the function f. In particular, we require the computation time of the clients and the communication complexity of the protocol to depend at most polylogarithmically in the circuit-size of f.[1] The computation time of the server S is allowed to be polynomial in the circuit-size of f.

We prove security of a cloud-assisted MPC protocol in the Ideal/Real paradigm described in Section 2.4.

3.1.2 Construction Overview

The starting point of our cloud-assited MPC construction is the blueprint of Gentry for how to achieve MPC from FHE [Gen09a]. Gentry's blueprint runs in 4 phases:

1. Parties run a general MPC protocol to obtain a public key pk and an evaluation key ek of an FHE scheme, as well as secret shares $\mathsf{sk}_1, \ldots, \mathsf{sk}_N$ of a corresponding secret key sk.

[1] We remark that we achieve complete independence when considering security against semi-malicious adversaries. It is only in the malicious setting that we must relax the definition to allow for a polylogarithmic factor in the circuit size of f.

2. Each party encrypts their input m_i under public key pk and broadcasts the resulting ciphertext c_i.

3. Each party individually and locally performs the homomorphic evaluation $c \stackrel{\mathsf{def}}{=} \mathsf{Eval}(\mathsf{ek}, C, c_1, \ldots, c_N)$, and

4. Parties run a general MPC protocol to decrypt c using their shares sk_i.

We optimize this construction by taking advantage of specific properties of the FHE scheme used to perform the homomorphic computation. We make three observations.

1. Our first observation is that the first step can be simplified if the FHE scheme has certain key homomorphic properties. This allows the parties to sample their own key pairs $(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)$ and then combine their public keys into a joint public key pk and their evaluation keys into a joint evaluation key ek, such that the individual secret keys sk_i's are secret shares of a valid secret key for pk and ek.

2. Our second observation is that the final decryption step can also be simplified if the FHE scheme has distributed decryption. This allows the parties to jointly decrypt c without incurring the expense of a general MPC protocol.

3. Finally, in our construction the server alone performs the homomorphic computation and broadcasts the result to all parties. This way, the computation of all parties except the server is independent of the size of the circuit being computed. Compactness of the FHE scheme guarantees that the communication complexity of the protocol is independent of the size of the circuit.

   In the malicious setting, the server must additionally prove to the clients that it carried out the computation correctly. This proof and its verification make the communication complexity of the protocol and the computation of all parties depend in the circuit size, but only polylogarithmically.

In Chapter 4, we show that the RLWE-based FHE scheme described in Section 2.7 is key homomorphic and has distributed decryption. We further give a formal description of our construction, and prove its security.

3.2 On-the-Fly MPC

We now consider a variant of MPC that has the same efficiency requirements as cloud-assisted MPC, but allows for more flexibility in the computation of functions, and requires less communication between parties. In this setting, a *server* or *cloud* S stores the data of a large universe of parties, $P_1, \ldots, P_U$. The data of each party must remain private, but the server S can compute any joint function on the data of any subset $V \subseteq [U]$ of the parties. Furthermore, the server must be able to perform this computation with minimal participation from the parties in V, and no interaction at all from the rest of the parties (those in $[U] \backslash V$).

Just as in the setting of cloud-assisted MPC, the amount of data the parties in V interchange, as well as their computation time, must be independent of the complexity of the function. But we further require that the communication complexity of the protocol and the computation of each party be independent of the size of the universe, U.

Finally, the computation should remain secure even when the server is untrusted. We formalize this idea below.

3.2.1 Definition

For a class $\mathcal{C}$ of functions with at most U inputs, an *on-the-fly multiparty protocol* Π for $\mathcal{C}$ is a protocol between U interactive Turing Machines $P_1, \ldots, P_U$, called *parties* or *clients*, and a *server* S, also an interactive Turing Machine. An on-the-fly MPC protocol consists of two phases:

Offline or Storage Phase: In the offline or storage phase, all parties $P_1, \ldots, P_U$ send (an encoding of) their inputs to the server S for storage. This phase is performed before a function $F \in \mathcal{C}$ and a computing set V are chosen. We require no interaction between parties; all communication must take place between a party and the server. Indeed, we assume parties are oblivious to other parties' existence during this phase.

Online Phase: The online phase begins once F is chosen, together with a subset V of inputs on which F will be evaluated. Only the server and parties in V participate in the online phase. After the function is selected, the server ignores all offline messages from non-computing parties (those in $[U] \backslash V$).

We require that for all inputs $\vec{x} = (x_1, \ldots, x_U)$ and all functions $F \in \mathcal{C}$, if F is an N-input function, then for all ordered subsets $V \subseteq [U]$ such that $|V| = N$, the output of Π in an execution where P_i is given x_i as input (the server S does not receive an input) and where F and V are chosen for the computation, is $y \stackrel{\mathsf{def}}{=} F((x_i)_{i \in V})$.

Efficiency Requirements. In the offline phase, the communication complexity of the protocol as well as the computation of all parties $P_1, \ldots, P_U$ and the server S can only depend on the security parameter κ and the size of their inputs.

In the online phase, we have the same efficiency requirements as in cloud-assisted MPC: we require the communication complexity of the protocol, as well as the computation time of parties in V to be *(essentially) independent* of the complexity of the function f. In particular, we require the computation time of the clients in V and the communication complexity of the online phase to depend at most polylogarithmically in the circuit-size of F.[2] The computation time of the server S is allowed to be polynomial in the circuit-size of F.

We further require that the computation of all parties (both in the offline and the online phases) be independent of the size of the universe U, but allow it to depend on N, the size of the computing set V.

Security. We prove security of a cloud-assisted MPC protocol in the Ideal/Real paradigm described in Section 2.4. Since the server ignores messages from parties outside V in the online phase, we assume w.l.o.g. that an adversary only corrupts computing parties (parties in V) and possibly the server S.

[2] We remark that we achieve complete independence when considering security against semi-malicious adversaries. It is only in the malicious setting that we must relax the definition to allow for a polylogarithmic factor in the circuit size of F.

3.2.2 Construction Overview

Recall once more Gentry's blueprint for constructing MPC from FHE [Gen09a]:

1. Parties run a general MPC protocol to obtain a public key pk and an evaluation key ek of an FHE scheme, as well as secret shares $\mathsf{sk}_1, \ldots, \mathsf{sk}_N$ of a corresponding secret key sk.

2. Each party encrypts their input m_i under public key pk and broadcasts the resulting ciphertext c_i.

3. Each party individually and locally performs the homomorphic evaluation $c \stackrel{\mathsf{def}}{=} \mathsf{Eval}(\mathsf{ek}, C, c_1, \ldots, c_N)$, and

4. Parties run a general MPC protocol to decrypt c using their shares sk_i.

In Chapter 4, we showed that if the above blueprint is instantiated using a key-homomorphic threshold FHE scheme, there is no need to run generic MPC protocols for joint key generation (Step 1) and decryption (Step 3). This yielded an efficient cloud-assisted MPC protocol.

Once again, we use Gentry's blueprint as a starting point for our on-the-fly MPC protocol. However, we make two important changes.

1. As in our construction of cloud-assisted MPC described in Section 3.1, the server alone performs the homomorphic computation and broadcasts the result to all parties. This way, the computation of all parties except the server is independent of the size of the circuit being computed. Compactness of the FHE guarantees that the communication complexity of the protocol is also independent of the size of the circuit. And once again, in the malicious setting both the communication complexity and the computation of all parties except the server does depend in the circuit size, but only polylogarithmically.

2. More importantly: in Gentry's blueprint, parties must interact in Step 1 before they are able to encrypt their inputs in Step 2. This is problematic in the setting of on-the-fly MPC, where the parties must send their inputs to the server for storage in an offline phase *before* they know what function will be computed and with what other parties the computation will take place. In particular, parties cannot interact with each other before they encrypt their inputs.

   Our solution is to remove Step 1 altogether. Instead of obtaining joint public and evaluation keys, each party P_i samples its own key tuple $(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)$ and encrypts its input m_i under its own public key pk_i. Of course, this presents a problem in the homomorphic evaluation in Step 3: the ciphertexts $c_1, \ldots, c_N$ are now encrypted under different public keys, a case that is not immediately handled by standard FHE.

   To this end, we define *multikey fully homomorphic encryption*. Intuitively, multikey FHE allows evaluating any circuit on ciphertexts that might be encrypted under *different* public keys. Furthermore, we show how to construct multikey FHE for any number of keys, N, based on the (modified) NTRU encryption scheme described in Section 2.8 [HPS98, SS11b].

   For more details, we refer the reader to Chapter 5 and Chapter 6. In Chapter 5, we give a formal definition of multikey FHE and a formal description of the NTRU-based multikey FHE construction. In Chapter 6, we formally describe of our construction of on-the-fly MPC and prove its security.

Chapter 4
Cloud-Assisted MPC from Threshold FHE

In this chapter, we show how to construct cloud-assisted MPC from threshold FHE: fully homomorphic encryption that is key-homomorphic and has distributed decryption. We start by defining key homomorphism and distributed decryption, and show that the FHE scheme described in Section 2.7 has these properties. We then give a construction of cloud-assisted MPC using this FHE scheme, and prove its security against semi-malicious adversaries. We then discuss how the protocol can be modified in order to guarantee security against malicious adversaries.

4.1 Threshold FHE

In this section, we show that the FHE scheme described in Section 2.7 is a threshold FHE scheme. However, before we are able to delve into key-homomorphism and distributed decryption, we must first show that ciphertexts from this scheme can be rerandomized. This is necessary in order to guarantee that evaluated ciphertexts do not leak information about the operations that were performed, since these operations will sometimes depend on secret keys.

4.1.1 Ciphertext Rerandomization

We show that the FHE from Section 2.7 has rerandomizable ciphertexts: there exists an algorithm ReRand such that for every polynomial-sized circuit C, given the public key pk and ciphertext $\mathbf{c} := \mathsf{Eval}(\mathsf{ek}, C, \mathbf{c}_1, \ldots, \mathbf{c}_\ell)$, where $\mathbf{c}_1, \ldots, \mathbf{c}_\ell$ are ciphertexts encrypting plaintexts $m_1, \ldots, m_\ell$ under pk, outputs a randomization $\widehat{\mathbf{c}}$ of $\mathbf{c}$ that is indistinguishable from a *fresh* encryption of $y \stackrel{\mathsf{def}}{=} C(m_1, \ldots, m_\ell)$, even given $\mathbf{c}_1, \ldots, \mathbf{c}_\ell$. Essentially, this shows that a rerandomized ciphertext does not leak any information about the circuit C.

As was observed by Gentry [Gen09a], this can be done by adding to $\mathbf{c}$ an encryption of 0 with noise that is super-polynomially larger than the noise in $\mathbf{c}$. To this end, we let Enc_χ denote the encryption algorithm that samples noise elements from distribution χ, and define $\mathsf{ReRand}_\chi(\mathsf{pk}, \mathbf{c}) \stackrel{\mathsf{def}}{=} \mathbf{c} + \mathsf{Enc}_\chi(\mathsf{pk}, 0)$. Then the following lemma directly follows from Corollary 2.6.6.

Lemma 4.1.1. *Suppose the magnitude of the noise in ciphertext $\mathbf{c}$ is bounded by B. If $\chi = \overline{D}_{\mathbb{Z}^n, r}$ for $r \geq 2^{\omega(\log \kappa)} B$, then the noise distributions of $\mathsf{ReRand}_\chi(\mathsf{pk}, \mathbf{c})$ and a fresh encryption $\mathsf{Enc}_\chi(\mathsf{pk}, y)$ are statistically close.*

Corollary 4.1.2. *Let $\mathbf{c}_1, \ldots, \mathbf{c}_\ell$ be ciphertexts with noise bounded by B, and let $\mathbf{c} := \mathsf{Eval}(\mathsf{ek}, C, \mathbf{c}_1, \ldots, \mathbf{c}_\ell)$. If $\chi = \overline{D}_{\mathbb{Z}^n, r}$ for $r \geq 2^{\omega(\log \kappa)} B$, then the noise distributions of $\mathsf{ReRand}_\chi(\mathsf{pk}, \mathbf{c})$ and a fresh encryption $\mathsf{Enc}_\chi(\mathsf{pk}, y)$ are statistically close.*

Looking ahead, parties will need to rerandomize ciphertexts when creating a joint evaluation key and when performing distributed decryption, so as not to leak any information about the homomorphic operations that were computed (which depend on their secret key).

4.1.2 Key Homomorphism

The FHE scheme from Section 2.7 is *key homomorphic*: it allows combining public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_N$ into a combined public key pk, evaluation keys $\mathsf{ek}_1, \ldots, \mathsf{ek}_N$ into a combined evaluation key ek, and secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_N$ into a combined secret key sk, such that if for all i, pk_i and ek_i are valid public and evaluation keys for secret key sk_i, then pk and ek are valid public and evaluation keys for sk. Formally, if for all $i \in [N]$, $(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)$ is in the support of Keygen, then $(\mathsf{pk}, \mathsf{sk}, \mathsf{ek})$ is also in the support of Keygen.

Combining Public Keys. Let $a \leftarrow R_q$, and let $\mathsf{pk}_1 = p_1 = [as_1 + x_1]_q$ and $\mathsf{pk}_2 = p_2 = [as_2 + x_2]_q$ be two public keys corresponding to two (independent) secret keys $\mathsf{sk}_1 = (1, s_1)$ and $\mathsf{sk}_2 = (1, s_2)$, but sharing the *same* parameter a. We have:

$$[p_1 + p_2]_q = [a(s_1 + s_2) + (x_1 + x_2)]_q$$

Thus, the public key $\mathsf{pk} := [\mathsf{pk}_1 + \mathsf{pk}_2]_q$ is a valid public key for the secret key $\mathsf{sk} := \mathsf{sk}_1 + \mathsf{sk}_2$. Furthermore, bounds on the size of the combined secret key and the combined error can be obtained by summing the bounds for the individual secret keys and error terms, respectively. Generalizing this observation to multiple multi-level keys yields the following lemma.

Lemma 4.1.3. *Let $N \in \mathbb{N}$ and for $i \in [N]$, let pk_i be a public key for secret key sk_i. Then, $\mathsf{pk} \stackrel{\mathsf{def}}{=} [\mathsf{pk}_1 + \ldots + \mathsf{pk}_N]_q$ is a valid public key for secret key $\mathsf{sk} \stackrel{\mathsf{def}}{=} \mathsf{sk}_1 + \ldots + \mathsf{sk}_N$. Furthermore, if for all i, $\|\mathsf{sk}_i\|_\infty < B_{\mathsf{sk}}$ and the error in pk_i is bounded by B_χ, then $\|\mathsf{sk}\|_\infty < NB_{\mathsf{sk}}$ and the error in pk is bounded by NB_χ.*

Combining Evaluation Keys. We describe how to compute a joint evaluation key in the simplied case of two parties and a single multiplication. We then generalize the solution to the setting of multiple multi-level keys.

Let $\mathsf{sk}_1 = (\mathbf{s}_1, \mathbf{t}_1)$ and $\mathsf{sk}_2 = (\mathbf{s}_2, \mathbf{t}_2)$ with $\mathbf{s}_i = (1, s_i)$ and $\mathbf{t}_i = (1, t_1)$, be two secret keys with joint secret key $\mathsf{sk} = (\mathbf{s}, \mathbf{t}) \stackrel{\mathsf{def}}{=} (\mathbf{s}_1 + \mathbf{s}_2, \mathbf{t}_1 + \mathbf{t}_2)$. Let $s \stackrel{\mathsf{def}}{=} s_1 + s_2$ and $t \stackrel{\mathsf{def}}{=} t_1 + t_2$ so that $\mathbf{s} = (1, s)$ and $\mathbf{t} = (1, t)$. Additionally, let $(\boldsymbol{\gamma}_1, \boldsymbol{\zeta}_1, \ldots)$ and $(\boldsymbol{\gamma}_2, \boldsymbol{\zeta}_2, \ldots)$ be evaluation keys for sk_1 and sk_2, respectively, and let $(\boldsymbol{\alpha}_1, \cdot), (\boldsymbol{\alpha}_2, \cdot)$ be their corresponding parameters.

We wish to compute pseudo-encryptions of $\mathsf{Pow}(s)$ and $\mathsf{Pow}(s^2)$ under $\mathbf{t}$. This is possible with some additional information: define $\mathbf{b}_{1,2}, \mathbf{b}_{2,1}, \mathbf{c}_1, \mathbf{c}_2$ as follows: for $i \neq j$, let $\mathbf{b}_{i,j} \stackrel{\mathsf{def}}{=} [\boldsymbol{\alpha}_j t_i + \mathbf{e}_{i,j}]_q$ for $\mathbf{e}_{i,j} \leftarrow \chi^{\lceil \log q \rceil}$, and define

$$\begin{aligned}\mathbf{v}_j &\stackrel{\mathsf{def}}{=} \left[\boldsymbol{\gamma}_j + \mathbf{b}_{i,j}\right]_q \\ &= [\boldsymbol{\alpha}_j(t_1 + t_2) + \mathbf{e}_j + \mathsf{Pow}(s_j)]_q = [\boldsymbol{\alpha}_j t + \mathbf{e}_j + \mathsf{Pow}(s_j)]_q\end{aligned}$$

where the equality holds for some for "small" $\mathbf{e}_j$. We have,

$$[\mathbf{v}_1 + \mathbf{v}_2]_q = [(\boldsymbol{\alpha}_1 + \boldsymbol{\alpha}_2)t + (\mathbf{e}_1 + \mathbf{e}_2) + \mathsf{Pow}(s_1 + s_2)]_q$$

Therefore $(\boldsymbol{\gamma}, \boldsymbol{\zeta})$, where $\boldsymbol{\gamma} \stackrel{\mathsf{def}}{=} [\mathbf{v}_1 + \mathbf{v}_2]_q$ and $\boldsymbol{\zeta} \stackrel{\mathsf{def}}{=} [-(\boldsymbol{\alpha}_1 + \boldsymbol{\alpha}_2)]_q$, is a pseudo-encryption of $\mathsf{Pow}(s)$ under key $\mathbf{t}$.

Moreover, given that $s^2 = (s_1+s_2)^2 = s_1^2 + s_1 s_2 + s_2 s_1 + s_2^2$, we can compute pseudo-encryptions of $\mathsf{Pow}\left(s^2\right)$ by computing, for every $i, j \in \{1, 2\}$:

$$\mathbf{c}_{i,j} := [s_i \mathbf{v}_j]_q = [(s_i \boldsymbol{\alpha}_j)\, t + (s_i \mathbf{e}_j) + \mathsf{Pow}(s_i s_j)]_q$$

Then, $(\widetilde{\boldsymbol{\gamma}}, \widetilde{\boldsymbol{\zeta}})$, where $\widetilde{\boldsymbol{\gamma}} \stackrel{\mathsf{def}}{=} \left[\sum_{i=1}^{2}\sum_{j=1}^{2} \mathbf{c}_{i,j}\right]_q$, $\widetilde{\boldsymbol{\zeta}} \stackrel{\mathsf{def}}{=} \left[\sum_{i=1}^{2}\sum_{j=1}^{2} s_i \boldsymbol{\alpha}_j\right]_q$, is a pseudo-encryption of $\mathsf{Pow}\left(s^2\right)$ under key $\mathbf{t}$.

We can therefore let $\mathsf{ek} := \left(\boldsymbol{\gamma}, \boldsymbol{\zeta}, \widetilde{\boldsymbol{\gamma}}, \widetilde{\boldsymbol{\zeta}}\right)$ be the evaluation key for the combined secret key $\mathsf{sk} = (\mathbf{s}, \mathbf{t})$.

A few remarks are in order. First, it is easy to see that we can generalize the above discussion to the setting of multiple multi-level keys. Second, note that in constructing the joint evaluation key, we did not use the elements $(\widetilde{\boldsymbol{\gamma}}_i, \widetilde{\boldsymbol{\zeta}}_i)$ from the individual evaluation keys, since we obtained pseudo-encryptions of $\mathsf{Pow}\left(s_i^2\right)$ in a different manner (by computing $\mathbf{c}_{i,i}$). This means that in our protocol, parties will not need to compute this part of their individual secret keys. Finally, note that the joint evaluation key can be computed in a 2-round interactive protocol. In Round 1, each party P_j computes and broadcasts its evaluation key $\boldsymbol{\gamma}_i$ and the elements $\mathbf{b}_{i,j}$ for all $i \neq j$. In Round 2, each party P_j computes and broadcasts $\mathbf{c}_{i,j}$ for all i. With this information it is possible to locally compute the combined evaluation key ek.

However, note that the $\mathbf{c}_{i,j}$'s reveal information about s_j, and it is therefore not secure for party P_j to broadcast these values.[1] This can be prevented by rerandomizing each $\mathbf{c}_{i,j}$, i.e. by adding to it an encryption of 0 with super-polynomially larger noise, as explained above.

Lemma 4.1.4. *Let $N \in \mathbb{N}$ and for $i \in [N]$, let ek_i be a valid evaluation key for secret key sk_i. Then, ek, computed from sk_i and ek_i as described above, is a valid evaluation key for secret key $\mathsf{sk} \stackrel{\mathsf{def}}{=} \mathsf{sk}_1 + \ldots + \mathsf{sk}_N$. Furthermore, if for all i, $\|\mathsf{sk}_i\|_\infty < B_{\mathsf{sk}}$ and the error in ek_i is bounded by B_χ, then the error in ek is bounded by $N^2[2nB_{\mathsf{sk}}B_\chi + B_{\mathsf{rand}}]$, where B_{rand} is a bound on the noise added at randomization.*

4.1.3 Distributed Decryption

The FHE scheme from Section 2.7 also has distributed decryption: it is possible to decrypt a ciphertext under the combined public key pk by independently using the secret keys sk_i. More formally, given a ciphertext $\mathbf{c}$ encrypting a plaintext m under pk, it is possible to compute a decryption share μ_i using only sk_i such that no information is revealed about sk_i and combining the ciphertext $\mathbf{c}$ with the shares $\mu_1, \ldots, \mu_N$ yields the plaintext m. Our construction follows the work of Asharov et al. [AJW11, AJL+12]. We remark that distributed decryption for a similar LWE-based scheme was described by Bendlin and Damgård [BD10].

Let $\mathsf{sk}_1 = \mathbf{s}_1$ and $\mathsf{sk}_2 = \mathbf{s}_2$ be two secret keys, and let $\mathsf{pk}_1 = p_1$ and $\mathsf{pk}_2 = p_2$ be two corresponding public keys. Let $\mathsf{sk} = \mathbf{s} = \mathbf{s}_1 + \mathbf{s}_2$ be the combined secret key, and let $\mathsf{pk} = p = p_1 + p_2$ be

[1] For example, given $\mathbf{c}_{i,j}$ and $\mathbf{c}_{i,j'}$ for $j \neq j'$, it is possible to obtain (a multiple of) s_j by computing their GCD.

the combined public key. Furthermore, let $\mathbf{c} = (v, w)$ be an encryption of m under pk, so that $m = \left[\left\lfloor \frac{2}{q} \cdot [\langle \mathbf{c}, \mathbf{s} \rangle]_q \right\rceil\right]_2$.

If we define $\mu_i = ws_i$, then $[v - \mu_1 - \mu_2]_q = \langle \mathbf{c}, \mathbf{s} \rangle$, as desired. However, μ_i reveals information about the secret key s_i. This is avoided by rerandomizing the share: we define decryption shares $\mu_i \stackrel{\mathsf{def}}{=} ws_i + e_i$, where e_i is super-polynomially larger than the noise in ws_i.

As before, we can generalize this to the multikey setting.

Lemma 4.1.5. *Let $N \in \mathbb{N}$. For $i \in [N]$, let $(\mathsf{pk}_i, \mathsf{sk}_i, \cdot)$ be a valid key tuple, and let pk be the combined public key $\mathsf{pk} = \mathsf{pk}_1 + \ldots + \mathsf{pk}_N$. Let $\mathbf{c} = (v, w)$ be a ciphertext under public key pk, and let B be a bound on the magnitude of the noise in $\mathbf{c}$. Performing distributed decryption with shares $\mu_i = ws_i + e_i$, where $\|e_i\|_\infty < B_{\mathsf{rand}}$, is correct as long as $B + NB_{\mathsf{rand}} + 1 < q/4$.*

We remark that although each share $\mu_i = ws_i + e_i$ *looks like* a RLWE sample, we are unable to base the security of distributed decryption on the RLWE assumption, as is done in the work of Bendlin and Damgård [BD10]. This is because in our setting, the ciphertext $\mathbf{c} = (v, w)$ being decrypted is the output of a homomorphic evaluation (on possibly incorrectly distributed ciphertexts), and therefore w is not guaranteed to be uniformly distributed. The result of this is that the noise e_i added to the share must be much larger than is required for RLWE to be hard.

We further note that a previous version of this work [LTV11] relies on the RLWE assumption to prove the security of the decryption shares, and attempts to rerandomize the ciphertext $\mathbf{c}$ into a ciphertext $\widehat{\mathbf{c}}$ that is indistinguishable from a fresh encryption of y, in order to guarantee that w is uniformly random. The ciphertext $\widehat{\mathbf{c}}$ is computed by having each party contribute an encryption of 0 with super-polynomially larger noise, and adding these ciphertexts to $\mathbf{c}$. The hope is that if at least one party is honest, then it will sample the noise correctly and this honest noise will flood the noise in $\mathbf{c}$. However, it must not only flood the noise in $\mathbf{c}$, but also the noise of all the corrupted parties, which can be just as large. Moreover, the adversary sees $\mathbf{c}$ and all the encryptions of 0, so it can easily distinguish between $\widehat{\mathbf{c}}$ and a fresh encryption of y. We are unable to prove the security of this approach, and instead use the approach used in the work of Asharov et al. [AJW11, AJL+12], as described above.

4.2 The Basic Protocol

Now that we have shown the RLWE-based FHE described in Section 2.7 is a threshold FHE and has rerandomizable ciphertexts, we can describe our cloud-assisted MPC protocol. We first describe a protocol that is secure against (static) semi-malicious corruptions, and then show how to transform this protocol into one that is secure against (static) malicious corruptions (see Section 4.3). Both protocols assume a common reference string (CRS).

4.2.1 Overview

Our construction follows Gentry's blueprint [Gen09a] for how to construct MPC from FHE, but optimizes it by instantiating the FHE scheme with the RLWE-based scheme described in Section 2.7 and taking advantage of its key homomorphic, distributed decryption, and ciphertext rerandomization properties.

The protocol has 4 rounds:

Input: All parties and the server receives a common reference string (CRS) that contains parameters for their public keys and evaluation keys. Crucially, the CRS contains a *single* parameter vector $\left(a^{(0)}, \ldots, a^{(D)}\right)$ for *all* public keys. This ensures that a joint public key can be computed using the key homomorphic properties of the scheme. On the other hand, there is an evaluation-key parameter vector $\left(\boldsymbol{\alpha}_i^{(0)}, \ldots, \boldsymbol{\alpha}_i^{(D)}\right)$ *for each party* P_i. This is necessary to maintain security.[2]

Round 1: Each party samples its own key pair $(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)$ and broadcasts $(\mathsf{pk}_i, \mathsf{ek}_i)$ and the first round in the 2-round protocol for computing the combined evaluation key.

Round 2: Having received all parties' public keys, each party can now compute the combined public key pk, and encrypt its input under pk. It broadcasts this ciphertext, $\mathbf{c}_i$, as well as the second round in the 2-round protocol for computing the combined evaluation key.

Round 3: At this point, the server can compute the combined evaluation key ek and perform the homomorphic evaluation $\mathbf{c} = \mathsf{Eval}\left(\mathsf{ek}, C, \mathbf{c}_1, \ldots, \mathbf{c}_N\right)$.

Round 4: Having received $\mathbf{c}$, parties compute and broadcast their corresponding decryption shares.

Local computation: Once they obtain all decryption shares, the parties can combine these shares with the ciphertext $\mathbf{c}$ and obtain the output.

We now give a formal description of the protocol.

4.2.2 Formal Protocol

The protocol uses four different distributions: $\chi_{\text{SK}}, \chi_{\text{ENC}}, \chi_{\text{EK}}, \chi_{\text{DEC}}$. The distribution χ_{SK} is used to sample secret keys, χ_{ENC} is used by the encryption algorithm to sample noise for fresh ciphertexts, χ_{EK} is used for randomization in the computation of the joint evaluation key, and χ_{DEC} is used for rerandomization in distributed decryption. They are all (truncated) discrete Gaussians with different standard deviations so as to guarantee the rerandomization requirements. We defer describing the specific parameters until after the protocol description.

Input: All parties and the server receive as input the common reference string:

$$\mathsf{crs} = \left(a^{(0)}, \ldots, a^{(D)}, \left\{\boldsymbol{\alpha}_i^{(0)}, \ldots, \boldsymbol{\alpha}_i^{(D)}\right\}_{i \in [N]}\right)$$

All parties will use the coefficient vectors $\left(a^{(0)}, \ldots, a^{(D)}\right)$ to generate their public keys. Party P_i will use $\left(\boldsymbol{\alpha}_i^{(0)}, \ldots, \boldsymbol{\alpha}_i^{(D)}\right)$ to generate its evaluation key.

Round 1: For $i \in [N]$, party P_i:

[2] Otherwise, for each d and j, the elements $\mathbf{b}_{i,j}^{(d)}$ and $\boldsymbol{\gamma}_j^{(d)}$ would be RLWE samples with the same parameter $\boldsymbol{\alpha}^{(d)}$ and the same secret key $s_j^{(d)}$, which is not secure.

- Generates a key tuple using the CRS: For $d \in \{0, \ldots, D\}$, samples ring elements $s_i^{(d)}, x_i^{(d)} \leftarrow \chi_{\text{ENC}}$. Defines $\mathbf{s}_i^{(d)} = \left(1, s_i^{(d)}\right)$ and $p_i^{(d)} = \left[a^{(d)} s_i^{(d)} + x_i^{(d)}\right]_q$, and sets

$$\mathsf{sk}_i := \left(\mathbf{s}_i^{(0)}, \ldots, \mathbf{s}_i^{(D)}\right) \ , \ \mathsf{pk}_i := \left(p_i^{(0)}, \ldots, p_i^{(D)}\right)$$

For $d \in [D]$, samples $\boldsymbol{\varepsilon}_i^{(d)}, \widetilde{\boldsymbol{\varepsilon}}_i^{(d)} \leftarrow \chi_{\text{ENC}}^{\lceil \log q \rceil}$, and sets $\mathsf{ek}_i := \left\{\boldsymbol{\gamma}_i^{(d)}, \boldsymbol{\zeta}_i^{(d)}\right\}_{d \in [D]}$, where

$$\boldsymbol{\gamma}_i^{(d)} := \left[\boldsymbol{\alpha}_i^{(d)} \cdot s_i^{(d)} + \boldsymbol{\varepsilon}_i^{(d)} + \mathsf{Pow}\left(s_i^{(d-1)}\right)\right]_q \in R_q^{\lceil \log q \rceil} \quad , \quad \boldsymbol{\zeta}_i^{(d)} := -\boldsymbol{\alpha}_i^{(d)}$$

Recall that the elements $\left\{\widetilde{\boldsymbol{\gamma}}_i^{(d)}, \widetilde{\boldsymbol{\zeta}}_i^{(d)}\right\}_{d \in [D]}$ in the individual evaluation keys are not needed to compute a combined evaluation key, and thus do not need to be computed at all.

- Generates ring elements to be used in creating the joint evaluation key: For all $j \in [N]$, $j \neq i$, and $d \in [D]$, samples $\mathbf{e}_{i,j}^{(d)} \leftarrow \chi_{\text{ENC}}^{\lceil \log q \rceil}$ and computes

$$\mathbf{b}_{i,j}^{(d)} := \left[\boldsymbol{\alpha}_j^{(d)} s_i^{(d)} + \mathbf{e}_{i,j}^{(d)}\right]_q$$

- Broadcasts $(\mathsf{pk}_i, \mathsf{ek}_i)$ and $\left\{\mathbf{b}_{i,j}^{(d)}\right\}_{j \in [N], j \neq i, d \in [D]}$

Round 2: For $i \in [N]$, party P_i:

- Computes the joint public key: For $d \in \{0, \ldots, D\}$, computes $p^{(d)} := p_1^{(d)} + \ldots + p_N^{(d)}$ and sets $\mathsf{pk} := \left(p^{(0)}, \ldots, p^{(D)}\right)$.
- Generates an encryption of its input m_i under the joint public key pk: Samples $r_i, e_i, e_i' \leftarrow \chi_{\text{ENC}}$, and computes the ciphertext $\mathbf{c}_i := (v_i, w_i) \in R_q^2$, where:

$$v_i := \left[p^{(0)} r_i + e_i + \left\lfloor \frac{q}{2} \right\rfloor m_i\right]_q \in R_q \quad , \quad w_i := \left[-a^{(0)} r_i + e_i'\right]_q \in R_q$$

- Generates ring elements to be used in creating the joint evaluation key: For all $j \in [N], d \in [D]$, samples $\boldsymbol{\rho}_j^{(d)}, \boldsymbol{\xi}_j^{(d)}, \boldsymbol{\xi}_j'^{(d)} \leftarrow \chi_{\text{EK}}^{\lceil \log q \rceil}$ and computes

$$\mathbf{v}_j^{(d)} := \left[\boldsymbol{\gamma}_j^{(d)} + \sum_{\substack{k=1 \\ k \neq j}}^{N} \mathbf{b}_{k,j}^{(d)}\right]_q \quad , \quad \mathbf{w}_j^{(d)} := -\boldsymbol{\alpha}_j^{(d)}$$

$$\mathbf{c}_{i,j}^{(d)} := \left[s_i^{(d-1)} \mathbf{v}_j^{(d)} + p^{(d)} \boldsymbol{\rho}_j^{(d)} + \boldsymbol{\xi}_j^{(d)}\right]_q \quad , \quad \mathbf{z}_{i,j}^{(d)} := \left[-s_i^{(d-1)} \boldsymbol{\alpha}_j^{(d)} - a^{(d)} \boldsymbol{\rho}_j^{(d)} + \boldsymbol{\xi}_j'^{(d)}\right]_q$$

Note that each pair $\left(\mathbf{c}_{i,j}^{(d)}, \mathbf{z}_{i,j}^{(d)}\right)$ is rerandomized by adding an encryption of 0 with noise that we will set to be super-polynomially larger. This ensures that the pair is statistically close to a fresh encryption of $s_i^{(d-1)} s_j^{(d-1)}$ and therefore does not reveal any information about $s_i^{(d)}$.

- Broadcasts $\mathbf{c}_i$ and $\left\{\mathbf{c}_{i,j}^{(d)}, \mathbf{z}_{i,j}^{(d)}\right\}_{j\in[N],d\in[D]}$

Round 3: The server S:

- Computes the joint evaluation key: Sets $\mathsf{ek} := \left\{\boldsymbol{\gamma}^{(d)}, \boldsymbol{\gamma}^{(d)}, \widetilde{\boldsymbol{\gamma}}^{(d)}, \widetilde{\boldsymbol{\zeta}}^{(d)}\right\}_{d\in[D]}$, where for $d \in [D]$:

$$\boldsymbol{\gamma}^{(d)} := \left[\sum_{j=1}^{N} \mathbf{v}_j^{(d)}\right]_q \quad , \quad \boldsymbol{\zeta}^{(d)} := \left[\sum_{j=1}^{N} \mathbf{w}_j^{(d)}\right]_q$$

$$\widetilde{\boldsymbol{\gamma}}^{(d)} := \left[\sum_{i=1}^{N}\sum_{j=1}^{N} \mathbf{c}_{i,j}^{(d)}\right]_q \quad , \quad \widetilde{\boldsymbol{\zeta}}^{(d)} := \left[\sum_{i=1}^{N}\sum_{j=1}^{N} \mathbf{z}_{i,j}^{(d)}\right]_q$$

  and the elements $\left\{\mathbf{v}_j^{(d)}, \mathbf{w}_j^{(d)}\right\}_{j\in[N],d\in[D]}$ are computed as in Round 2.
- Performs the homomorphic evaluation: $\mathbf{c} := \mathsf{Eval}(\mathsf{ek}, C, \mathbf{c}_1, \ldots, \mathbf{c}_N)$.
- Broadcasts $\mathbf{c}$.

Round 4: For $i \in [N]$, party P_i:

- Computes its decryption share: Parses $\mathbf{c} = (v, w)$ and samples $e_i \leftarrow \chi_{\text{DEC}}$. Computes:

$$\mu_i := w \cdot s_i^{(D)} + e_i \in R_q$$

- Broadcasts μ_i.

Local Computation: For $i \in [N]$, party P_i combines all the decryption shares to decrypt the evaluated ciphertext: Computes

$$\mu := \left[\left\lfloor \frac{2}{q}\left[v - \sum_{j=1}^{N} \mu_j\right]_q \right\rceil\right]_2$$

4.2.3 Choice of Parameters

The following set of parameters simultaneously ensures that:

- RLWE is hard.
- rerandomization is successful in both the creation of the joint evaluation key and in distributed decryption.
- distributed decryption is correct.

Given $N, D \in \mathbb{N}$, choose security parameter κ and set n such that

$$n = \omega\left(D \log n + \log D + D \log N + D \cdot \log^2 \kappa\right)$$

Note that the definition of n is circular. Although there is no analytical solution for such equation, programmatical solutions are available in terms of Lambert's W function or the generalized log function [Kal01]. Further set:

$$q = \Theta(n^{3D+6} D N^{D+6} \cdot 2^{(D+6)\cdot \log^2 \kappa}) \quad , \quad r_{\text{SK}} = r_{\text{ENC}} = 2^{\log^2 \kappa}$$

$$r_{\text{EK}} = n^2(N+1)2^{3\log^2 \kappa} \quad , \quad r_{\text{DEC}} = \Theta\left(n^{3D+5} D N^{D+5} \cdot 2^{(D+5)\cdot \log^2 \kappa}\right)$$

For $i \in \{\text{SK}, \text{ENC}, \text{EK}, \text{DEC}\}$, we let distribution $\chi_i = \overline{D}_{\mathbb{Z}, r_i}$, and observe that χ_i is B_i-bounded, for

$$B_{\text{SK}} = B_{\text{ENC}} = \sqrt{n} \cdot 2^{\log^2 \kappa} \ , \ B_{\text{EK}} = \sqrt{n} \cdot n^2(N+1)2^{3\log^2 \kappa} \ , \ B_{\text{DEC}} = \Theta\left(n^{3D+6} D N^{D+5} \cdot 2^{(D+5)\cdot \log^2 \kappa}\right)$$

Security of $\mathsf{RLWE}_{\phi, q, \chi_{\text{ENC}}}$ (and therefore security of the FHE and of our protocol) is based on the hardness of the $2^{\omega(\log n)} \cdot (q/r_{\text{ENC}})$-approximate R-SVP problem. With our setting of parameters, we have:

$$\begin{aligned} 2^{\omega(\log n)} \cdot (q/r_{\text{ENC}}) =& 2^{\omega(\log n)} \cdot \frac{\Theta\left(n^{3D+6} D N^{D+6} \cdot 2^{(D+6)\cdot \log^2 \kappa}\right)}{2^{\log^2 \kappa}} \\ \leq& 2^{\omega(\log n)} \cdot 2^{O(D \log n + \log D + D \log N + D \cdot \log^2 \kappa)} = \qquad 2^{\omega(\log n)} \cdot 2^{o(n)} \end{aligned}$$

This corresponds to solving the R-SVP problem to within a sub-exponential factor, which is believed to be hard.

We now bound the error in different elements in the protocol, and show that rerandomization requirements are met and that distributed decryption will be correct. The error of each $\mathbf{v}_j^{(d)}$ is bounded by $(N+1)B_{\text{ENC}}$. Therefore, the error in $s_j^{(d-1)}\mathbf{v}_j^{(d)}$ is bounded by $n(N+1)B_{\text{SK}}B_{\text{ENC}}$. We have set r_{EK} to be larger by a super-polynomial factor, as required for rerandomization.

The error of each $\mathbf{c}_{i,j}$ is bounded by $n(N+1)B_{\text{SK}}B_{\text{ENC}} + nNB_{\text{ENC}}B_{\text{EK}} + nNB_{\text{SK}}B_{\text{EK}} + B_{\text{EK}} = O(nNB_{\text{ENC}}B_{\text{EK}})$. This means that the error in the evaluation key $\boldsymbol{\gamma}, \widetilde{\boldsymbol{\gamma}}$ is bounded by $O(nN^3 B_{\text{ENC}}B_{\text{EK}})$. By Theorem 2.7.1, and the fact that the secret key size is bounded by NB_{SK}, this means that the error of $\mathbf{c}$ is bounded by:

$$\begin{aligned} & O\left((n^3 N B_{\text{SK}})^D \left(nB_{\text{ENC}}^2 + nNB_{\text{ENC}}B_{\text{SK}} + n^2N^2B_{\text{SK}}^2 + n\left\lceil \log q \right\rceil \cdot (nN^3 B_{\text{ENC}}B_{\text{EK}})\right)\right) \\ &= O\left((n^3 N B_{\text{SK}})^D \left(n\left\lceil \log q \right\rceil \cdot (nN^3 B_{\text{ENC}}B_{\text{EK}})\right)\right) \\ &= O\left((n^3 N \cdot 2^{\log^2 \kappa})^D \left(n^4 N^4 \left\lceil \log q \right\rceil 2^{4\log^2 \kappa}\right)\right) \\ &= O\left(n^{3D+4} N^{D+4} \left\lceil \log q \right\rceil 2^{(D+4)\log^2 \kappa}\right) = O\left(n^{3D+5} D N^{D+5} 2^{(D+4)\log^2 \kappa}\right) \end{aligned}$$

We have set r_{DEC} to be larger than the error of $\mathbf{c}$ by a super-polynomial factor, in order to guarantee success in rerandomizing the decryption shares.

All that is left is to show that decryption will be correct. The error in decryption is the error in $\mathbf{c}$ plus the sum of the errors in the decryption shares. Thus, the decryption error is bounded by:

$$O\left(n^{3D+5}DN^{D+5}2^{(D+4)\log^2\kappa}\right) + N\sqrt{n}\cdot\Theta\left(n^{3D+5}DN^{D+5}2^{(D+5)\log^2\kappa}\right)$$
$$= O\left(n^{3D+6}DN^{D+6}2^{(D+5)\log^2\kappa}\right)$$

Decryption is correct as long as this error is smaller than $q/4$; we guarantee this by setting $q = \Omega\left(n^{3D+6}DN^{D+6}2^{(D+5)\log^2\kappa}\right)$.

4.2.4 Security Against Semi-Malicious Adversaries

Theorem 4.2.1. *Let f be an N-input function. Let C be a poly-sized circuit computing f, and let $\mathcal{F}$ be the ideal functionality computing f. Assuming the hardness of $\mathsf{RLWE}\phi, q, \chi_{\text{ENC}}$, the protocol described above securely implements $\mathcal{F}$ against (static) semi-malicious adversaries.*

Proof of Theorem 4.2.1: We prove correctness, performance, and security.

Correctness: Correctness follows from Lemma 4.1.3, Lemma 4.1.4, Lemma 4.1.5, and our choice of parameters described above.

Performance: By compactness of evaluation, we know that $\mathbf{c}$ is independent of the size of C. This means that the computation of the clients, as well as the communication complexity of the protocol, is independent of the size of C.

Security: We show security for the case when the server is corrupted; the case when the server is honest is analogous. Let $\mathcal{A}$ be a real-world semi-malicious adversary corrupting $t < N$ clients and the server. Let $T \subsetneq [N]$ be the set of corrupted clients. We assume, without loss of generality, that whenever the protocol instructs a corrupted party to sample form a B-bounded distribution, $\mathcal{A}$ chooses an element that is B-bounded, though not necessarily from the correct distribution[3].

We construct a simulator $\mathcal{S}^{\text{SM}}$ as follows. The simulator receives the inputs of the corrupted parties, $\{m_i\}_{i\in T}$. It creates the CRS honestly, and runs $\mathcal{A}$ on the inputs $\{m_i\}_{i\in T}$ and the CRS. It simulates the messages for all honest parties in the protocol execution with $\mathcal{A}$. It does this by first fixing an arbitrary honest party $h \in \overline{T}$. For all honest parties P_j such that $j \neq h$, the simulator creates all messages honestly, except the encryption of the input m_j, which it simulates with random ring elements. For the honest party P_h, the simulator chooses all messages at random, and uses P_h's decryption share μ_h to fix the outcome of the computation. Details follow.

Round 1: For all $i \in \overline{T}, i \neq h$, the simulator computes $\mathsf{pk}_i, \mathsf{ek}_i, \left\{\mathbf{b}_{i,j}^{(d)}\right\}_{j\neq i}$ honestly. For party P_h it samples uniform ring elements instead: For $d \in \{0,\ldots,D\}$, samples $p_h^{(d)} \leftarrow R_q$ and sets $\mathsf{pk}_h := \left(p_h^{(0)},\ldots,p_h^{(D)}\right)$. For $d \in [D]$, samples $\boldsymbol{\gamma}_h^{(d)} \leftarrow R_q^{\lceil \log q\rceil}$, sets $\boldsymbol{\zeta}_h^{(d)} := -\boldsymbol{\alpha}_h^{(d)}$

[3]This could be ensured by changing the protocol to explicitly mention sampling a B-bounded element. We choose not to do this for the sake of clarity in the description of the protocol.

and sets $\mathsf{ek}_h := \left\{\boldsymbol{\gamma}_h^{(d)}, \boldsymbol{\zeta}_h^{(d)}\right\}_{d\in[D]}$. For all $j \in [N], j \neq h, d \in [D]$, it samples $\mathbf{b}_{h,j}^{(d)} \leftarrow R_q$ uniformly at random.

For all honest parties $i \in \overline{T}$, the simulator sends $\left(\mathsf{pk}_i, \mathsf{ek}_i, \left\{\mathbf{b}_{i,j}^{(d)}\right\}_{j\neq i, d\in[D]}\right)$ to $\mathcal{A}$ as P_i's broadcast message.

When it receives messages from $\mathcal{A}$ for all corrupted parties, it reads from $\mathcal{A}$'s witness tape the secret keys $\left\{\mathsf{sk}_k^{(d)}\right\}_{k\in T}$ for all corrupted parties P_k.

Round 2: For all $i \in \overline{T}$, instead of computing an encryption of input m_i (which it doesn't know), the simulator samples random ring elements: $\mathbf{c}_i \leftarrow R_q^2$.

Furthermore, for $i \in \overline{T}, i \neq h$, the simulator computes $\left\{\mathbf{c}_{i,j}^{(d)}, \mathbf{z}_{i,j}^{(d)}\right\}$ honestly, but chooses $\mathbf{c}_{h,j}^{(d)}, \mathbf{z}_{h,j}^{(d)} \leftarrow R_q^{\lceil \log q \rceil}$ for all $j \in [N], d \in [D]$.

For all $i \in \overline{T}$, the simulator sends $\left(\mathbf{c}_i, \left\{\mathbf{c}_{i,j}^{(d)}, \mathbf{z}_{i,j}^{(d)}\right\}_{j\in[N], d\in[D]}\right)$ to $\mathcal{A}$ as P_i's broadcast message.

When it receives messages from $\mathcal{A}$ for all corrupted parties, it reads from $\mathcal{A}$'s witness tape the inputs $\{\overline{m}_k\}_{k\in T}$. The simulator sends these inputs to the trusted functionality $\mathcal{F}$ and receives the output y.

Round 3: The simulator receives ciphertext $\mathbf{c} = (v, w)$ from $\mathcal{A}$ as the server's broadcast message.

Round 4: For $k \in T$, let $\mathsf{sk}_k = \left(\mathbf{s}_k^{(0)}, \ldots, \mathbf{s}_k^{(D)}\right)$ with $\mathbf{s}_k^{(d)} = \left(1, s_k^{(D)}\right)$, be P_k's secret key that the simulator obtained at the end of Round 1. For all $i \in \overline{T}, i \neq h$, the simulator computes the decryption share μ_i honestly, and uses μ_h to *fix* the output of the decryption to y:

$$\mu_h = v - y - \sum_{\substack{i=1 \\ i\neq h}}^{N} w s_i^{(D)} + e$$

where $e \leftarrow \chi_{\text{DEC}}$.

Output: The simulator receives the output of the corrupted parties from $\mathcal{A}$, and returns these as its output.

We now show that $\text{REAL}_{\Pi^{\text{SH}},\mathcal{A}}(\vec{x}) \overset{c}{\approx} \text{IDEAL}_{\mathcal{F},\mathcal{S}^{\text{SM}}}(\vec{x})$ via a series of hybrids.

Hybrid 0: This is the real-world execution of the protocol.

Hybrid 1: In this hybrid, we change how the share μ_h is computed:

$$\mu_h = v - y - \sum_{\substack{i=1 \\ i\neq h}}^{N} w s_i^{(D)} + e$$

where $e \leftarrow \chi_{\text{DEC}}$, and where for $k \in T$, $\mathsf{sk}_k = \left(\mathbf{s}_k^{(0)}, \ldots, \mathbf{s}_k^{(D)}\right)$ with $\mathbf{s}_k^{(d)} = \left(1, s_k^{(D)}\right)$, is P_k's secret key.

We claim that the view of $\mathcal{A}$ in Hybrid 0 is statistically close to the view of $\mathcal{A}$ in Hybrid 1. Everything is the same in both hybrids except the share μ_h. Furthermore, by correctness of evaluation-key combination and homomorphic evaluation, we know that $\mathbf{c}$ is an encryption of y with some noise ε. Then, in Hybrid 1,

$$\mu_h = ws^{(D)} + \varepsilon + y - y - \sum_{\substack{i=1 \\ i \neq h}}^{N} ws_i^{(D)} + e = ws_h^{(D)} + \varepsilon + e$$

Since $e \leftarrow \chi_{\text{DEC}}$ and we chose r_{DEC} to be super-polynomially larger than the magnitude of ε, by Corollary 2.6.6 the distributions of μ_h in Hybrid 0 and Hybrid 1 are statistically close.

Hybrid 2: In this hybrid, we change how the elements $\left(\mathbf{c}_{h,j}^{(d)}, \mathbf{z}_{h,j}^{(d)}\right)$ are computed. They are fresh encryptions of $s_h^{(d-1)} s_j^{(d-1)}$ with noise from χ_{EK}. For elements $\overline{\boldsymbol{\rho}}_j^{(d)}, \overline{\boldsymbol{\xi}}_j^{(d)}, \overline{\boldsymbol{\xi}}_j'^{(d)} \leftarrow \chi_{\text{EK}}$:

$$\mathbf{c}_{h,j}^{(d)} = \left[p^d \overline{\boldsymbol{\rho}}_j^{(d)} + \overline{\boldsymbol{\xi}}_j^{(d)} + s_h^{(d-1)} s_j^{(d-1)}\right]_q \quad , \quad \mathbf{z}_{h,j}^{(d)} = \left[-a^d \overline{\boldsymbol{\rho}}_j^{(d)} + \overline{\boldsymbol{\xi}}_j'^{(d)}\right]_q$$

By Lemma 4.1.1, we know that Hybrid 1 and Hybrid 2 are statistically close.

Hybrid $3.(d, z)$ **for** $d = D, \ldots, 0$ **and** $z \in \{0, 1\}$**:** In Hybrid $3.(d, 0)$, we change the protocol so that the elements $b_{h,j}^{(d)}$, the public key $p_h^{(d)}$, and the evaluation key $\boldsymbol{\gamma}_h^{(d)}$ are now chosen uniformly at random. Then, in Hybrid $3.(d, 1)$, we change the protocol once more so that the elements $\mathbf{c}_{h,j}^{(d)}, \mathbf{z}_{h,j}^{(d)}$ are now chosen uniformly at random as well. We must do this in two steps because to argue that the elements $\mathbf{c}_{h,j}^{(d)}$ are pseudorandom, we must first argue that the public key $p_h^{(d)}$ is pseudorandom, and to argue that the public key $p_h^{(d)}$ is pseudorandom, we must first argue that the elements $\mathbf{c}_{h,j}^{(d+1)}$ are pseudorandom.

In Hybrid $3.(d, 0)$, we sample:

$$\left\{\mathbf{b}_{h,j}^{(d)} \leftarrow R_q^{\lceil \log q \rceil}\right\}_{j \neq h} \quad , \quad p_h^{(d)} \leftarrow R_q^n \quad , \quad \boldsymbol{\gamma}_h^{(d)} \leftarrow R_q^n$$

In Hybrid $3.(d, 1)$, we further sample:

$$\left\{\mathbf{c}_{h,j}^{(d)}, \mathbf{z}_{h,j}^{(d)} \leftarrow R_q^{\lceil \log q \rceil}\right\}_{j \in [N]}$$

We define Hybrid $3.(D+1, 1)$ to be the same as Hybrid 2. We first claim that the view of $\mathcal{A}$ in Hybrid $3.(d, 0)$ and Hybrid $3.(d+1, 1)$ are computationally indistinguishable by the RLWE assumption with secret $s_h^{(d)}$. Indeed, notice that the only places in the protocol that $s_h^{(d)}$ is used is as a RLWE secret when constructing these elements, and "encrypted" in the elements $\boldsymbol{\gamma}_h^{(d+1)}$ and $\mathbf{c}_{h,j}^{(d+1)}$. Our hybrid approach guarantees that

in both Hybrid 3.$(d,0)$ and Hybrid 3.$(d+1,1)$ the elements $\boldsymbol{\gamma}_h^{(d+1)}$ and $\mathbf{c}_{h,j}^{(d+1)}$ are both uniformly random (or in the case of $d = D$, don't exist).

We now claim that Hybrid 3.$(d,0)$ and Hybrid 3.$(d,1)$ are computationally indistinguishable by the RLWE assumption with secret $\overline{\boldsymbol{\rho}}_j^{(d)}$ (see Fact 2.6.8). In both hybrids, $p_h^{(d)}$ is uniformly random and thus the combined public key $p^{(d)}$ is uniformly random as well. Furthermore, the element $-a^{(d)}$ is guaranteed to be uniformly random.

Hybrid 4: In this hybrid, we change how the ciphertexts $\mathbf{c}_i$ are computed for all honest parties. Instead of encrypting the real input m_i, they are sampled as random ring elements:

$$\left\{\mathbf{c}_i \leftarrow R_q^2\right\}_{i\in\overline{T}}$$

We claim that Hybrid 4 is computationally indistinguishable from Hybrid 3 by the RLWE assumption with secret $\mathbf{r} = (r_i)_{i\in T}$ (see Fact 2.6.8). This follows from the fact that $p_h^{(0)}$ is uniformly random and thus, the combined public key $p^{(0)}$ is also uniformly random. Furthermore, the element $-a^{(0)}$ is guaranteed to be uniformly random.

Hybrid 4 is precisely the outcome of running the simulator $\mathcal{S}^{\text{SM}}$. We conclude that $\text{REAL}_{\Pi^{\text{SH}},\mathcal{A}}(\vec{x}) \overset{c}{\approx} \text{IDEAL}_{\mathcal{F},\mathcal{S}^{\text{SM}}}(\vec{x})$.

□

4.3 Achieving Security Against Malicious Adversaries

To guarantee security against malicious adversaries, we apply the AJW compiler [AJW11, AJL+12] (see Section 2.4.2), to the protocol described in Section 4.2 which is secure against semi-malicious corruptions. This entails requiring each party and the server to prove in zero-knowledge, at every round, that its messages in that round are well-formed and consistent with the protocol transcript so far.

However, verifying the server's proof in Round 3 requires time proportional to the size of the circuit C. Therefore, a simple application of the compiler eliminates the performance guarantees of the cloud-assisted protocol. We therefore replace the server's proof with a *succinct argument*. We offer several solutions, each offering a unique set of advantages and drawbacks.

Verifying the Server's Computation. The server needs to convince the parties that "$c = \mathsf{Eval}(C, (c_1, \mathsf{pk}_1, \mathsf{ek}_1), \ldots, (c_N, \mathsf{pk}_N, \mathsf{ek}_N))$", or in other words, that a deterministic circuit of size $\mathsf{poly}(|C|, \kappa)$ accepts. For any uniform circuit C (i.e., computable by a $\mathsf{poly}(\kappa)$-time Turing machine), the following offer $\mathsf{poly}(\kappa, \log(|C|))$ communcation and verification efficiency.[4]

1. Use the argument system of Kilian [Kil92, Kil95], yielding *interactive* 4-round verification. It relies on expensive PCPs.

[4]For any given family of C, $|C| = \mathsf{poly}(\kappa)$, and thus, $\mathsf{poly}(\kappa, \log(|C|)) = \mathsf{poly}(\kappa)$; but the degree of this polynomial depends on the circuit family.

2. Use the succinct non-interactive arguments (SNARGs and SNARKs) of Micali [Mic94], Bitansky et al. [BCCT12, BCCT13] or Goldwasser et al. [GLR11] (see Section 2.3). These are non-interactive[5] but are secure only in the random oracle model [BR93] (in the case of CS proofs) or hold in the standard model but require a non-falsifiable assumption [Nao03]. Some variants rely on PCPs, PIR or FHE.

If we allow a pre-processing phase in which clients are able to perform computation proportional to the size of the function to be computed, we have two more solutions:

4. Use verifiable computation protocols in the pre-processing model (e.g. [GGP10, CKV10, AIK10]). They rely on FHE.

5. Use SNARGs/SNARKs where the CRS depends on the circuit to be computed or where its size is at least as big as the computation, e.g. [Gro10, Lip12, GGPR13, PHGR13, Lip13]. These are based on non-falsifiable assumptions.

Finally, in case that the evaluation circuit is in logspace-uniform **NC**, we have another alternative:

6. Use the argument system of Goldwasser et al. [GKR08] for a 1-round solution[6]. It relies on PIR.

[5] In our protocol, each party can run Gen in Step 1 and send the vrs to the server in that step. Or in the case of CS proofs, where only a description of a hash function is required, this can be added to the CRS of the protocol.

[6] The protocol has 2 rounds, but (as in the case of SNARGs and SNARKs) the first round is a challenge that is independent of the language and the statement, and can therefore be precomputed by the clients in Round 1 or 2 of our protocol. Each challenge can only be used for one proof, so the client must send a new challenge in each execution of the protocol.

Chapter 5
Multikey FHE

Having constructed cloud-assisted MPC for all efficiently computable functions, we now focus on constructing a protocol that satisfies the more stringent requirements of on-the-fly MPC. As mentioned earlier, the main building block in this construction is multikey FHE: fully homomorphic encryption that allows homomorphic evaluation on ciphertexts encrypted under different and independent keys. In this chapter, we formally define multikey FHE and show a construction for any number of keys based on the NTRU encryption scheme [HPS98, SS11b] described in Section 2.8. We also show that any FHE scheme is inherently multikey for a constant number of keys (in the security parameter), and that the Brakerski-Vaikuntanathan scheme [BV11b, BGV12] is somewhat homomorphic for a logarithmic number of keys.

5.1 Definition

To formally define multikey fully homomorphic encryption, we introduce a parameter N, which is the number of distinct keys that the scheme can handle; all algorithms will depend polynomially on N. This is similar to the definition of leveled homomorphic encryption from [BGV12] (see Definition 2.5.3), but we note that in our definition, the algorithms depend on N but are independent of the depth of circuits that the scheme can evaluate. Thus, we consider schemes that are "leveled" with respect to the number of keys N, but fully homomorphic ("non-leveled") with respect to the circuits that are evaluated. The construction of multikey FHE schemes that are not leveled with respect to the number of keys (i.e., where all algorithms are independent of N) remains an open problem.

Finally, we note that to guarantee semantic security, decryption requires all corresponding secret keys.

Definition 5.1.1 (Multikey $\mathcal{C}$-Homomorphic Encryption)**.** *Let $\mathcal{C}$ be a class of circuits. A family $\left\{\mathcal{E}^{(N)} = (\mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})\right\}_{N>0}$ of algorithms is* multikey $\mathcal{C}$-homomorphic *if for all integers $N > 0$, $\mathcal{E}^{(N)}$ has the following properties:*

- $(\mathsf{pk}, \mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{Keygen}(1^\kappa)$*: For a security parameter κ, outputs a public key* pk*, a secret key* sk *and a (public) evaluation key* ek*.*
- $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$*: Given a public key* pk *and message m, outputs a ciphertext c.*
- $m := \mathsf{Dec}\,(\mathsf{sk}_1, \ldots, \mathsf{sk}_N, c)$*: Given N secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_N$ and a ciphertext c, outputs a message m.*
- $c := \mathsf{Eval}(C, (c_1, \mathsf{pk}_1, \mathsf{ek}_1), \ldots, (c_\ell, \mathsf{pk}_\ell, \mathsf{ek}_\ell))$*: Given a (description of) a boolean circuit C along with ℓ tuples $(c_i, \mathsf{pk}_i, \mathsf{ek}_i)$, each comprising of a ciphertext c_i, a public key pk_i, and an evaluation key ek_i, outputs a ciphertext c.*

We require absence of decryption failures and compactness of ciphertexts. Formally: for every circuit $C \in \mathcal{C}$, all sequences of N key tuples $\left\{\left(\mathsf{pk}'_j, \mathsf{sk}'_j, \mathsf{ek}'_j\right)\right\}_{j\in[N]}$ each of which is in the support of $\mathsf{Keygen}(1^\kappa)$, all sequences of ℓ key tuples $\{(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)\}_{i\in[\ell]}$ each of which is in $\left\{\left(\mathsf{pk}'_j, \mathsf{sk}'_j, \mathsf{ek}'_j\right)\right\}_{j\in[N]}$, and all plaintexts $(m_1, \ldots, m_\ell)$ and ciphertexts $(c_1, \ldots, c_\ell)$ such that c_i is in the support of $\mathsf{Enc}(\mathsf{pk}_i, m_i)$, Eval satisfies the following properties:

Correctness: *Let $c := \mathsf{Eval}(C, (c_1, \mathsf{pk}_1, \mathsf{ek}_1), \ldots, (c_\ell, \mathsf{pk}_\ell, \mathsf{ek}_\ell))$. Then $\mathsf{Dec}\left(\mathsf{sk}'_1, \ldots, \mathsf{sk}'_N, c\right) = C(m_1, \ldots, m_\ell)$.*[1]

Compactness: *Let $c := \mathsf{Eval}(C, (c_1, \mathsf{pk}_1, \mathsf{ek}_1), \ldots, (c_\ell, \mathsf{pk}_\ell, \mathsf{ek}_\ell))$. There exists a polynomial P such that $|c| \leq P(\kappa, N)$. In other words, the size of c is independent of ℓ and $|C|$. Note, however, that we allow the evaluated ciphertext to depend on the number of keys, N.*

Definition 5.1.2 (Multikey Fully Homomorphic Encryption)**.** *A family of encryption schemes $\left\{\mathcal{E}^{(N)} = (\mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})\right\}_{N>0}$ is* multikey fully homomorphic *if it is multikey $\mathcal{C}$-homomorphic for the class $\mathcal{C}$ of all circuits.*

Semantic security of a multikey FHE follows directly from the semantic security of the underlying encryption scheme in the presence of the evaluation key ek. This is because given ek, the adversary can compute Eval himself. Note that taking $N = 1$ in Definition 5.1.1 and Definition 5.1.2 yield the standard definitions of $\mathcal{C}$-homomorphic and fully homomorphic encryption schemes (Definition 2.5.1 and Definition 2.5.2).

5.2 Multikey FHE for a Small Number of Keys

As a prelude to our main result in Section 5.3, we show that multikey homomorphic encryption for a small number of keys can be easily achieved. In particular, we show that any (standard) FHE can be converted into a multikey FHE for a constant number of keys, $N = O(1)$. Furthermore, we show that the Brakerski-Vaikuntanathan (ring-based) FHE [BV11b] is multikey homomorphic for a logarithmic number of keys, $N = O(\log \kappa)$. Unfortunately, once we introduce multiple keys we are unable to use either relinearization or squashing, and can therefore only obtain a somewhat homomorphic encryption scheme.

5.2.1 $O(1)$-Multikey FHE from any FHE

We show that any FHE scheme is inherently multikey for a constant number of keys, $N = O(1)$.[2] Let $\mathcal{E} = (\mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be an FHE scheme with message space $\{0,1\}$ and ciphertext space $\{0,1\}^\lambda$ where $\lambda = p(\kappa)$ for some polynomial $p(\cdot)$. For $x \in \{0,1\}^*$, define $x[i]$ to be the ith bit of x, and define $\widetilde{\mathsf{Enc}}$ to be the bit-wise encryption of x:

$$\widetilde{\mathsf{Enc}}(\mathsf{pk}, x) \stackrel{\mathsf{def}}{=} (\mathsf{Enc}(\mathsf{pk}, x[1]), \ldots, \mathsf{Enc}(\mathsf{pk}, x[|x|]))$$

[1] Note that correctness still holds even if the circuit C completely ignores all ciphertexts encrypted under a public key pk'_i, or if none of the original ciphertexts were encrypted under this key. In other words, using superfluous keys in the decryption process does not affect its correctness (as long as decryption uses at most N keys).

[2] The idea for this construction was originally suggested to us by an anonymous STOC 2012 reviewer. We include it in this dissertation and formally prove its correctness for the sake of completeness.

Furthermore, for any $k \in \mathbb{N}$, recursively define "onion" encryption and decryption:

$$\begin{aligned}
\mathsf{Enc}^*(\mathsf{pk}, x) &\stackrel{\mathsf{def}}{=} \mathsf{Enc}(\mathsf{pk}, x) \\
\mathsf{Enc}^*\left(\mathsf{pk}_1, \ldots, \mathsf{pk}_k, x\right) &\stackrel{\mathsf{def}}{=} \mathsf{Enc}^*\left(\mathsf{pk}_1, \ldots, \mathsf{pk}_{k-1}, \mathsf{Enc}\left(\mathsf{pk}_k, x\right)\right) \\
&\stackrel{\mathsf{def}}{=} \mathsf{Enc}\left(\mathsf{pk}_1, \mathsf{Enc}\left(\mathsf{pk}_2, \ldots, \mathsf{Enc}\left(\left(\mathsf{pk}_k, x\right)\right)\right)\right) \\
\mathsf{Dec}^*(\mathsf{sk}, x) &\stackrel{\mathsf{def}}{=} \mathsf{Dec}(\mathsf{sk}, x) \\
\mathsf{Dec}^*\left(\mathsf{sk}_1, \ldots, \mathsf{sk}_k, x\right) &\stackrel{\mathsf{def}}{=} \mathsf{Dec}^*\left(\mathsf{sk}_2, \ldots, \mathsf{pk}_k, \mathsf{Dec}\left(\mathsf{sk}_1, x\right)\right) \\
&= \mathsf{Dec}\left(\mathsf{sk}_k, \mathsf{Dec}\left(\mathsf{sk}_{k-1}, \ldots, \mathsf{Dec}\left(\mathsf{sk}_1, x\right)\right)\right)
\end{aligned}$$

We highlight two properties of "onion" encryption and decryption:

1. First, note that Enc^* and Dec^* satisfy correctness: if $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Keygen}(1^\kappa)$ for all $i \in [k]$, then for all $m \in \{0,1\}$:

$$\mathsf{Dec}^*\left(\mathsf{sk}_1, \ldots, \mathsf{sk}_k, \mathsf{Enc}^*\left(\mathsf{pk}_1, \ldots, \mathsf{pk}_k, m\right)\right) = m$$

2. Second, note that the bit-size of the ciphertext $\mathsf{Enc}^*\left(\mathsf{pk}_1, \ldots, \mathsf{pk}_k, m\right)$ is λ^k. Recall that the ciphertext space of Enc is $\{0,1\}^\lambda$ and $\lambda = p(\kappa)$ for some polynomial $p(\cdot)$.

Construction Overview. We now give an overview of the construction. Given N ciphertexts $c_i \leftarrow \mathsf{Enc}\left(\mathsf{pk}_i, m_i\right)$ encrypting plaintext m_i under key pk_i, for all $i \in [N]$, it is possible to homomorphically compute "onion" ciphertexts:

$$z_i \approx \mathsf{Enc}^*\left(\mathsf{pk}_1, \ldots, \mathsf{pk}_N, m_i\right)$$

This is done by homomorphically evaluating the function $\mathsf{Enc}^*\left(\mathsf{pk}_{i+1}, \ldots, \mathsf{pk}_N, \ \cdot\ \right)$ on ciphertext c_i. This outputs an onion encryption $\widetilde{z}_i \approx \mathsf{Enc}^*\left(\mathsf{pk}_i, \ldots, \mathsf{pk}_N, m_i\right)$. The ciphertext z_i can be obtained by onion encrypting $\widetilde{z}_i$ with the remaining keys: $z_i = \mathsf{Enc}^*\left(\mathsf{pk}_1, \ldots, \mathsf{pk}_{i-1}, \widetilde{z}_i\right)$

Once the ciphertexts $z_1, \ldots, z_N$ have been obtained, we can recursively perform homomorphic evaluations corresponding to the keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_N$ (in that order), to obtain a ciphertext:

$$c \approx \mathsf{Enc}^*\left(\mathsf{pk}_1, \ldots \mathsf{pk}_N, C\left(m_1, \ldots, m_N\right)\right)$$

By correctness of "onion" encryption, decrypting c can be easily achieved using "onion" decryption:

$$\mathsf{Dec}^*\left(\mathsf{sk}_1, \ldots, \mathsf{sk}_k, c\right) = C\left(m_1, \ldots, m_N\right)$$

However, recall that the size of each ciphertext z_i is $\lambda^N = p(\kappa)^N$ for some polynomial $p(\cdot)$. This means that the multikey homomorphic evaluation is efficient only if $N = O(1)$. Thus, this generic construction of multikey FHE from (standard) FHE allows only a constant number of keys.

Formal Description. We now give a formal description of the generic multikey construction, and prove its correctness. Let $\mathcal{E} = (\mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be an FHE scheme with message space $\{0,1\}$ and ciphertext space $\{0,1\}^\lambda$ where $\lambda = p(\kappa)$ for some polynomial $p(\cdot)$. Let Enc^* and Dec^* be the "onion" encryption and decryption algorithms described above.

- $\mathsf{GMK.Keygen}(1^\kappa)$: Run $\mathsf{Keygen}(1^\kappa)$.

- $\mathsf{GMK.Enc}(\mathsf{pk}, m)$: Run $\mathsf{Enc}(\mathsf{pk}, m)$.
- $\mathsf{GMK.Dec}\left(\mathsf{sk}_1, \ldots, \mathsf{sk}_N, c\right)$: Output $\mathsf{Dec}^*\left(\mathsf{sk}_1, \ldots, \mathsf{sk}_N, c\right)$.
- $\mathsf{GMK.Eval}\left(C, \left(c_1, \mathsf{pk}_1, \mathsf{ek}_1\right), \ldots, \left(c_N, \mathsf{pk}_N, \mathsf{ek}_N\right)\right)$: For $i \in [N]$, define

$$G_i(x) \stackrel{\mathsf{def}}{=} \mathsf{Enc}^*\left(\mathsf{pk}_{i+1}, \ldots, \mathsf{pk}_N, x \ ; \ r\right)$$

for some fixed and valid randomness r,[3] and recursively define

$$C^{(k)}\left(x_1, \ldots, x_N\right) \stackrel{\mathsf{def}}{=} \begin{cases} C\left(x_1, \ldots, x_N\right) & \text{for } k = N \\ \mathsf{Eval}\left(\mathsf{ek}_{k+1}, C^{(k+1)}, x_1, \ldots, x_N\right) & \text{for } k < N \end{cases}$$

For $i \in [N]$, compute

$$\widetilde{z}_i \stackrel{\mathsf{def}}{=} \mathsf{Eval}\left(\mathsf{ek}_i, G_i, c_i\right) \quad , \quad z_i \stackrel{\mathsf{def}}{=} \mathsf{Enc}^*\left(\mathsf{pk}_1, \ldots, \mathsf{pk}_{i-1}, \widetilde{z}_i\right)$$

and output the ciphertext $c \stackrel{\mathsf{def}}{=} \mathsf{Eval}\left(\mathsf{ek}_1, C^{(1)}, z_1, \ldots, z_N\right)$.

Theorem 5.2.1. *The encryption scheme* $\mathcal{E}_{\mathsf{GMK}} = (\mathsf{GMK.Keygen}, \mathsf{GMK.Enc}, \mathsf{GMK.Dec}, \mathsf{GMK.Eval})$ *is multikey fully homomorphic for* $N = O(1)$ *keys.*

Proof. To prove correctness of evaluation, we wish to prove that if $(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)$ is in the support of $\mathsf{GMK.Keygen}(1^\kappa) = \mathsf{Keygen}(1^\kappa)$ and $c_i \leftarrow \mathsf{GMK.Enc}\left(\mathsf{pk}_i, m_i\right) = \mathsf{Enc}\left(\mathsf{pk}_i, m_i\right)$, then

$$\mathsf{GMK.Dec}\left(\mathsf{sk}_1, \ldots, \mathsf{sk}_N, c\right) = \mathsf{Dec}^*\left(\mathsf{sk}_1, \ldots, \mathsf{sk}_N, c\right) = C\left(m_1, \ldots, m_N\right)$$

We first show that each z_i is a valid "onion" encryption of m_i. By correctness of evaluation with evaluation key ek_i, we know that

$$\mathsf{Dec}\left(\mathsf{sk}_i, \widetilde{z}_i\right) = G_i\left(m_i\right) = \mathsf{Enc}^*\left(\mathsf{pk}_{i+1}, \ldots, \mathsf{pk}_N, m_i \ ; \ r\right)$$

and by correctness of encryption, we conclude that

$$\mathsf{Dec}^*\left(\mathsf{sk}_i, \ldots, \mathsf{sk}_N, \widetilde{z}_i\right) = m_i \quad \text{and} \quad \mathsf{Dec}^*\left(\mathsf{sk}_1, \ldots, \mathsf{sk}_N, z_i\right) = m_i$$

We now make the following claim, which constitutes the bulk of the proof.

Claim 5.2.1.1. *For every* $k \in [N]$,

$$\mathsf{Dec}^*\left(\mathsf{sk}_1, \ldots, \mathsf{sk}_k, c\right) = C_k\left(z_1^{(k)}, \ldots, z_N^{(k)}\right)$$

where $z_i^{(k)} \stackrel{\mathsf{def}}{=} \mathsf{Dec}^*\left(\mathsf{sk}_1, \ldots, \mathsf{sk}_k, z_i\right)$.

In particular, for $k = N$, this claim implies:

$$\mathsf{Dec}^*\left(\mathsf{sk}_1, \ldots, \mathsf{sk}_N, c\right) = C^{(N)}\left(z_1^{(N)}, \ldots, z_N^{(N)}\right) = C\left(m_1, \ldots, m_N\right)$$

where the second equality follows from the fact that $C_N = C$ by definition, and the fact that $z_i^{(N)} = \mathsf{Dec}^*\left(\mathsf{sk}_1, \ldots, \mathsf{sk}_N, z_i\right) = m_i$, which we proved earlier.

It thus suffices to prove Claim 5.2.1.1 to conclude the proof of the theorem.

[3] We need to include the randomness in the definition because we want $G_i(x)$ to be a deterministic circuit with x as its sole input.

Proof. We prove Claim 5.2.1.1 by induction. The base case, $k = 1$, follows directly from correctness of evaluation and correctness of decryption:

$$\mathsf{Dec}^*\left(\mathsf{sk}_1, c\right) = C^{(1)}\left(\mathsf{Dec}\left(\mathsf{sk}_1, z_1\right), \ldots, \mathsf{Dec}\left(\mathsf{sk}_1, z_N\right)\right) = C^{(1)}\left(z_1^{(1)}, \ldots, z_N^{(1)}\right)$$

Now suppose that the claim holds for $k-1$; that is, suppose

$$\mathsf{Dec}^*\left(\mathsf{sk}_1, \ldots, \mathsf{sk}_{k-1}, c\right) = C^{(k-1)}\left(z_1^{(k-1)}, \ldots, z_N^{(k-1)}\right)$$

Decrypting both sides by sk_k yields:

$$\begin{aligned}
\mathsf{Dec}^*\left(\mathsf{sk}_1, \ldots, \mathsf{sk}_k, c\right) &= \mathsf{Dec}\left(\mathsf{sk}_k, C^{(k-1)}\left(z_1^{(k-1)}, \ldots, z_N^{(k-1)}\right)\right) \\
&= \mathsf{Dec}\left(\mathsf{sk}_k, \mathsf{Eval}\left(\mathsf{ek}_k, C^{(k)}, z_1^{(k-1)}, \ldots, z_N^{(k-1)}\right)\right) \\
&= C^{(k)}\left(\mathsf{Dec}\left(\mathsf{sk}_k, z_1^{(k-1)}\right), \ldots, \mathsf{Dec}\left(\mathsf{sk}_k, z_N^{(k-1)}\right)\right) \\
&= C^{(k)}\left(z_1^{(k)}, \ldots, z_1^{(k)}\right)
\end{aligned}$$

where the second-to-last equality follows from correctness of evaluation and correctness of decryption. This concludes the inductive step and the proof. □

□

5.2.2 $O(\log \kappa)$-Multikey FHE from Ring-LWE

We now show that the Brakerski-Vaikuntanathan FHE [BV11b] based on the RLWE assumption is multikey somewhat homomorphic for $N = O(\log \kappa)$ keys.

Recall from Section 2.7 that decryption in Regev encryption consists of computing the inner product $\langle \mathbf{c}, \mathbf{s} \rangle \pmod 2$, where $\mathbf{c}, \mathbf{s} \in R_q^2$ are the ciphertext and secret key, respectively. Brakerski and Vaikuntanathan [BV11b] generalize this to allow the ciphertext and secret key to grow in dimension. For $\mathbf{c}, \mathbf{s} \in R_q^d$, they define: $\mathsf{Dec}(\mathbf{s}, \mathbf{c}) = \langle \mathbf{c}, \mathbf{s} \rangle \pmod 2$. Homomorphic operations are then defined as follows:

- Given two *same-length* ciphertexts $\mathbf{c}_1$ and $\mathbf{c}_2$, output the ciphertext $\mathbf{c}_{\mathsf{add}} \stackrel{\mathsf{def}}{=} \mathbf{c}_1 + \mathbf{c}_2$ as an encryption of the *sum* of the underlying messages.

  The ciphertext $\mathbf{c}_{\mathsf{add}}$ is decryptable with the same secret key $\mathbf{s}$ since

  $$\langle \mathbf{c}_1 + \mathbf{c}_2, \mathbf{s} \rangle = \langle \mathbf{c}_1, \mathbf{s} \rangle + \langle \mathbf{c}_2, \mathbf{s} \rangle$$

- Given two ciphertexts $\mathbf{c}_1$ and $\mathbf{c}_2$ *of potentially different length*, output the ciphertext $\mathbf{c}_{\mathsf{mult}} \stackrel{\mathsf{def}}{=} \mathbf{c}_1 \otimes \mathbf{c}_2$ as the *product* of the underlying messages.

  The ciphertext $\mathbf{c}_{\mathsf{mult}}$ is now decryptable with the secret key $\mathbf{s} \otimes \mathbf{s}$ since

  $$\langle \mathbf{c}_1 \otimes \mathbf{c}_2, \mathbf{s} \otimes \mathbf{s} \rangle = \langle \mathbf{c}_1, \mathbf{s} \rangle \cdot \langle \mathbf{c}_2, \mathbf{s} \rangle$$

We can extend this to the multikey setting. Multiplication is trivial, but some changes are necessary in the case of addition.

- Given two *same-length* ciphertexts $\mathbf{c}_1$ and $\mathbf{c}_2$ decryptable with secret keys $\mathbf{s}_1, \mathbf{s}_2$ respectively, output the ciphertext $\mathbf{c}_{\mathsf{add}} \stackrel{\mathsf{def}}{=} (\mathbf{c}_1, \mathbf{c}_2)$ as an encryption of the *sum* of the underlying messages.

  The ciphertext $\mathbf{c}_{\mathsf{add}}$ is decryptable with the same secret key $(\mathbf{s}_1, \mathbf{s}_2)$ since

$$\langle (\mathbf{c}_1, \mathbf{c}_2), (\mathbf{s}_1, \mathbf{s}_2) \rangle = \langle \mathbf{c}_1, \mathbf{s}_1 \rangle + \langle \mathbf{c}_2, \mathbf{s}_2 \rangle$$

- Given two ciphertexts $\mathbf{c}_1$ and $\mathbf{c}_2$ decryptable with secret keys $\mathbf{s}_1, \mathbf{s}_2$ respectively, and *of potentially different length*, output the ciphertext $\mathbf{c}_{\mathsf{mult}} \stackrel{\mathsf{def}}{=} \mathbf{c}_1 \otimes \mathbf{c}_2$ as the *product* of the underlying messages.

  The ciphertext $\mathbf{c}_{\mathsf{mult}}$ is now decryptable with the secret key $\mathbf{s}_1 \otimes \mathbf{s}_2$ since

$$\langle \mathbf{c}_1 \otimes \mathbf{c}_2, \mathbf{s}_1 \otimes \mathbf{s}_2 \rangle = \langle \mathbf{c}_1, \mathbf{s}_1 \rangle \cdot \langle \mathbf{c}_2, \mathbf{s}_2 \rangle$$

Observe that each homomorphic operation (at most) doubles the size of the ciphertext. Starting with fresh ciphertexts of length 2, after $(N-1)$ operations (which can combine ciphertexts encrypted under at most N distinct keys), the size of both the ciphertext and the joint decryption key is 2^N. This is only feasible if $N = O(\log \kappa)$.

As shown in the work of Brakerski and Vaikuntanathan [BV11b], the scheme can evaluate circuits of depth $D < \varepsilon \log n - \log\log n + \Theta(1)$, where $q = 2^{n^\varepsilon}$ for constant $\varepsilon \in (0,1)$. Unfortunately, we do not know how to apply relinearization or squashing in the multikey setting, and are therefore not able to convert the resulting multikey scheme into a leveled or fully homomorphic one.

5.3 Multikey Somewhat Homomorphic Encryption for Any Number of Keys

We now turn to our main result in this chapter. In this section, we construct a multikey somewhat homomorphic encryption scheme based on the (modified) NTRU encryption scheme [HPS98, SS11b] described in Section 2.8. Unlike the schemes in Section 5.2, the scheme we describe in this section will be multikey for $N \approx n^\varepsilon$ keys, with constant $\varepsilon \in (0,1)$. In Section 5.4, we show how to convert the scheme into a multikey fully homomorphic scheme for $N \approx n^\varepsilon$ keys. By setting $n \approx N^{1/\varepsilon}$, we can construct a multikey FHE for any number of keys N, as long as N is known a-priori.

We begin by informally describing the multikey homomorphic properties of NTRU encryption and some of the problems encountered when trying to convert the scheme from Section 2.8 into a somewhat homomorphic one. We then show a formal description of our somewhat homomorphic scheme, formally prove its homomorphic properties, and discuss its security. In Section 5.4, we show how to convert this scheme into a fully homomorphic scheme.

5.3.1 Multikey Homomorphism

Recall from Section 2.8 that an NTRU key pair consists of ring elements (h, f) such that $h = \left[2gf^{-1}\right]_q$, where g, f are "small" ring elements sampled from a B-bounded distribution χ, and f^{-1} is the inverse of f in R_q. Further recall that an NTRU ciphertext has the form $c = [hs + 2e + m]_q$ for "small" elements s, e sampled from χ, and decryption computes $[fc]_q \pmod 2$.

Let (h_1, f_1) and (h_2, f_2) be two different NTRU public/secret key pairs, and let $c_1 \stackrel{\mathsf{def}}{=} [h_1 s_1 + 2e_1 + m_1]_q$ and $c_2 \stackrel{\mathsf{def}}{=} [h_2 s_2 + 2e_2 + m_2]_q$ be two ciphertexts, encrypted under public

keys h_1 and h_2, respectively. We show how to compute ciphertexts that decrypt to the sum and the product of the underlying plaintexts, m_1 and m_2. In particular, we show that the "ciphertexts" $c_{mult} \stackrel{\mathsf{def}}{=} c_1 \cdot c_2$ and $c_{add} \stackrel{\mathsf{def}}{=} c_1 + c_2$ can be decrypted to the product and the sum of m_1 and m_2 respectively, using the secret key $f_{12} \stackrel{\mathsf{def}}{=} f_1 \cdot f_2$.

To see this, note that

$$[f_1 f_2 (c_1 + c_2)]_q = [2f_1 f_2 e_1 + 2f_1 f_2 e_2 + 2f_2 g_1 s_1 + 2f_1 g_2 s_2 + f_1 f_2 (m_1 + m_2)]_q$$

$$\begin{aligned}[f_1 f_2 (c_1 \cdot c_2)]_q &= [4g_1 g_2 s_1 s_2 + 2g_1 s_1 f_2 (2e_2 + m_2) + 2g_2 s_2 f_1 (2e_1 + m_1) \ + \\ &\qquad 2f_1 f_2 (e_1 m_2 + e_2 m_1 + 2e_1 e_2) + f_1 f_2 (m_1 m_2)]_q \\ &= m_1 \cdot m_2 \pmod 2\end{aligned}$$

Since $f_1 \equiv f_2 \equiv 1 \pmod 2$, we can conclude that as long as there is no reduction modulo q,

$$[f_1 f_2 (c_1 + c_2)]_q \pmod 2 = m_1 + m_2 \pmod 2$$

$$[f_1 f_2 (c_1 \cdot c_2)]_q \pmod 2 = m_1 \cdot m_2 \pmod 2$$

In other words, the "joint secret key" $f_{12} \stackrel{\mathsf{def}}{=} f_1 f_2$ can be used to decrypt $c_{\mathsf{add}} = [c_1 + c_2]_q$ and $c_{\mathsf{mult}} = [c_1 \cdot c_2]_q$. We can extend this argument to any combination of operations, with ciphertexts encrypted under multiple public keys.

Of course, the error in the ciphertexts will grow with the number of operations performed (as with all known fully homomorphic encryption schemes). Thus, correctness of decryption will only hold for a limited number of operations. We can show that the scheme can correctly evaluate circuits of depth roughly $\varepsilon \log(n)$ when $q = 2^{n^\varepsilon}$ and $B = \mathsf{poly}(n)$.

Problems in Multikey Decryption. An astute reader will have observed that in order to successfully decrypt a ciphertext that was the result of a homomorphic evaluation, we must know the circuit that was evaluated. For example, to decrypt $c_1^2 + c_2$ we need to multiply by $f_1^2 f_2$, whereas to decrypt $c_1 + c_2^2$ we need to multiply by $f_1 f_2^2$. This is unsatisfactory.

Furthermore, consider what happens when we add or multiply two ciphertexts c, c' that are themselves a result of homomorphic evaluation. Suppose, for example, that $c = c_1 c_2$ and $c' = c_2 c_3$, where c_i is encrypted under h_i for $i \in \{1, 2, 3\}$. We know c can be decrypted using $f_1 f_2$ and c' can be decrypted using $f_2 f_3$. Thus, we know that

$$[f_1 f_2 \cdot c]_q = 2e + f_1 f_2 m \quad , \quad \left[f_2 f_3 \cdot c'\right]_q = 2e' + f_2 f_3 m'$$

for some messages m and m' and error terms e and e'. Following the discussion above, we can see that $c + c'$ can be decrypted using the key $f_1 f_2 f_3$:

$$\left[f_1 f_2 f_3 \cdot (c + c')\right]_q = \left[f_3 (f_1 f_2 \cdot c) + f_1 (f_2 f_3 \cdot c')\right]_q = 2(f_3 e + f_1 e') + f_1 f_2 f_3 (m + m')$$

In general, given a ciphertext c encrypted under a set of keys K, and c' encrypted under a set of keys K', their sum can be decrypted using the product of the keys in the union $K \cup K'$. We

note that the absolute magnitude of the coefficients of this product grows exponentially with the number of keys in $K \cup K'$, i.e. the total number of keys involved in the homomorphic computation.

Analogously, in the context of homomorphic multiplication, we need $f_1 f_2^2 f_3$ to decrypt $c \cdot c'$:

$$\left[f_1 f_2^2 f_3 \cdot (c \cdot c')\right]_q = \left[(f_1 f_2 \cdot c) \cdot (f_2 f_3 \cdot c')\right]_q = 2E_{\mathsf{mult}} + f_1 f_2^2 f_3 (m \cdot m')$$

where $E_{\mathsf{mult}} \stackrel{\mathsf{def}}{=} 2ee' + ef_2 f_3 m' + e' f_1 f_2 m$. This hints at the fact that the magnitude of the coefficients of the joint secret key needed to decrypt an evaluated ciphertext grows *exponentially with the degree of the evaluated circuit* (and not just with the number of keys involved, as in the case of addition). Indeed, after M multiplications, the joint secret key needed to decrypt the evaluated ciphertext will be the product of M polynomials, and the magnitude of the coefficients of this product will be exponential in M.

Our Solution. To solve the above problems, we use *relinearization* (also known as *key-switching*), a technique first introduced by Brakerski and Vaikuntanathan [BV11a]. Informally, we introduce a (public) evaluation key ek that will be output by the Keygen algorithm. Every time we multiply ciphertexts that share a key f_i, we will use the evaluation key to ensure that we only need f_i, and not f_i^2, to decrypt the new ciphertext. This ensures two things.

1. First, it ensures that to decrypt a ciphertext c^*, we only need to multiply it by *one* copy of each secret key, making decryption independent of the circuit used to produce c^*.
2. Second, it ensures that the size of the joint secret key needed to decrypt the new ciphertext depends only on the number of keys N, and not on the degree of the circuit C that was evaluated.

Though we are able to eliminate the dependence (of the magnitude of the coefficients of the joint secret key) on the degree of the circuit, we remark that we do not succeed in eliminating the exponential dependence on N, the number of keys – indeed, this is the reason why our solution will eventually assume an a-priori upper bound on N.

5.3.2 Formal Description

We present a formal description of our multikey somewhat homomorphic encryption scheme based on the (modified) NTRU encryption scheme [HPS98, SS11b] described in Section 2.8.

- $\mathsf{SH.Keygen}(1^\kappa)$: Sample $f', g \leftarrow \chi$ and set $f := 2f' + 1$ so that $f \equiv 1 \pmod 2$. If f is not invertible in R_q, resample f'; otherwise let f^{-1} be the inverse of f in R_q. Set

$$\mathsf{pk} \stackrel{\mathsf{def}}{=} h := \left[2gf^{-1}\right]_q \in R_q \quad , \quad \mathsf{sk} \stackrel{\mathsf{def}}{=} f \in R$$

  Sample $\widetilde{\mathbf{s}}, \widetilde{\mathbf{e}} \leftarrow \chi^{\lceil \log q \rceil}$ and compute $\mathsf{ek} \stackrel{\mathsf{def}}{=} \left[h\widetilde{\mathbf{s}} + 2\widetilde{\mathbf{e}} + \mathsf{Pow}(f)\right]_q \in R_q^{\lceil \log q \rceil}$. Output the key tuple $(\mathsf{pk}, \mathsf{sk}, \mathsf{ek})$.

- $\mathsf{SH.Enc}(\mathsf{pk}, m)$: Sample $s, e \leftarrow \chi$. Output the ciphertext $c := hs + 2e + m \in R_q$.

- $\mathsf{SH.Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_N, c)$: Parse $\mathsf{sk}_i = f_i$ for $i \in [N]$. Compute $\mu = [f_1 \cdots f_N \cdot c]_q \in R_q$ and output $m := \mu \pmod 2$.

- $\mathsf{SH.Eval}(C, (c_1, \mathsf{pk}_1, \mathsf{ek}_1), \ldots, (c_\ell, \mathsf{pk}_\ell, \mathsf{ek}_\ell))$: We show how to evaluate an ℓ-variate boolean circuit $C : \{0,1\}^\ell \to \{0,1\}$ of depth D. To this end, we show how to homomorphically add and multiply two elements in $\{0,1\}$. Given two ciphertexts c, c', we assume that we also have a set of distinct public keys associated with each ciphertext.[4] We will denote these lists by K, K', respectively. The public-key set of a fresh encryption is simply the set $\{\mathsf{pk}\}$ containing the public key under which it was encrypted. For convenience, in our analysis we sometimes assume that the set contains the indices of the public keys instead of the keys themselves.

  - Given two ciphertexts c and c' with corresponding public-key sets K and K', output the ciphertext
  $$c_{\mathsf{add}} = \left[c + c'\right]_q \in R_q$$
  as an encryption of the *sum* of the underlying messages. Output the set $K_{\mathsf{add}} = K \cup K'$ as its corresponding public-key set.
  - Given two ciphertexts c and c' with corresponding public-key sets K and K', compute ciphertext $c_0 = [c \cdot c']_q \in R_q$.
    * If $K \cap K' = \emptyset$, let $c_{\mathsf{mult}} = c_0$.
    * Otherwise, let $K \cap K' = \left\{\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_t}\right\}$. For $j \in [t]$, compute $c_j = \left[\left\langle \mathsf{Bit}\left(c_{j-1}\right), \mathsf{ek}_{i_j}\right\rangle\right]_q$, and let $c_{\mathsf{mult}} = c_t$ at the end of the iteration.

    In either case, output c_{mult} as an encryption of the *product* of the underlying messages, and output the set $K_{\mathsf{mult}} = K \cup K'$ as its corresponding public-key set.

    For a set $S \subseteq [N]$, let $f_S \stackrel{\mathsf{def}}{=} \prod_{i \in S} f_i$. Note that the ciphertext c_0 can be decrypted to $m \cdot m'$ with the "joint" secret key $f_K f_{K'}$, which contains the term $f_{i_1}^2 \ldots f_{i_t}^2$. The goal of relinearization is to convert it into a ciphertext that decrypts to the same message under the secret key
    $$f_K f_{K'} \left(\prod_{j \in K \cap K'} f_j \right)^{-1} = f_{K \cup K'}$$
    which replaces the term $f_{i_1}^2 \ldots f_{i_t}^2$ with the term $f_{i_1} \ldots f_{i_t}$.

We first show that the scheme works correctly as advertised:

Lemma 5.3.1. *If $q = 2^{n^\varepsilon}$ for $\varepsilon \in (0,1)$ and χ is a B-bounded distribution for $B = \mathsf{poly}(n)$, then the encryption scheme $\mathcal{E}_{\mathsf{SH}} = (\mathsf{SH.Keygen}, \mathsf{SH.Enc}, \mathsf{SH.Dec}, \mathsf{SH.Eval})$ described above is multikey homomorphic for $N = O\left(n^\delta\right)$ keys and circuits of depth $D < (\varepsilon - \delta)\log n - \log\log n - O(1)$.*

Proof. We define the (multikey) error of a ciphertext c with corresponding public-key set K to be $\mu \stackrel{\mathsf{def}}{=} [f_K \cdot c]_q$. We start by showing that the magnitude of the error coefficients does not grow too much after a homomorphic evaluation.

Claim 5.3.1.1. *Let c, c' be ciphertexts encrypting m and m', respectively, and suppose that the magnitude of their error coefficients is bounded by $E < q/2$. Then c_{add} and c_{mult} correctly decrypt to $m + m'$ and $m \cdot m'$, respectively, and their error coefficients are bounded by $(nB)^{2N} E^2$.*

[4] That is, we assume each set contains distinct public keys, but the intersection of any two sets might not be empty.

Proof. Let c, c' be encryptions of m, m', respectively, with corresponding public-key sets K, K'. We know that for some $e, e' \in R$ we have:

$$[f_K \cdot c]_q = 2e + m \quad , \quad \left[f_{K'} \cdot c'\right]_q = 2e' + m'$$

and $\|2e + m\|_\infty, \|2e' + m\|_\infty < E$. Then

$$\begin{aligned}[f_{K_{\mathsf{add}}} \cdot c_{\mathsf{add}}]_q &= \left[f_{K \cup K'} \cdot (c + c')\right]_q = \left[f_{K \setminus K'}(f_K \cdot c) + f_{K' \setminus K}(f_{K'} \cdot c')\right]_q \\ &= f_{K \setminus K'}(2e + m) + f_{K' \setminus K}(2e' + m')\end{aligned}$$

We can thus bound the magnitude of the coefficients of $[f_{K_{\mathsf{add}}} \cdot c_{\mathsf{add}}]_q$ by $2(nB)^N E < (nB)^{2N} E^2$, as desired. Furthermore, it easy to see that $[f_{K_{\mathsf{add}}} \cdot c_{\mathsf{add}}]_q \pmod 2 = m + m'$.

The multiplication case is more complex. Let $K \cap K' = \{i_1, \ldots, i_t\}$, as before. Define $F_0 \stackrel{\mathsf{def}}{=} f_K f_{K'}$, and for $j \in [t]$, define $F_j = F_{j-1} \cdot f_{i_j}^{-1}$. Then $F_r = f_{K \cup K'}$ is a simple product of the secret keys f_i, without any quadratic terms. We know that

$$[F_0 \cdot c_0]_q = [(f_K \cdot c)(f_{K'} \cdot c_K)]_q = (2e + m)(2e' + m')$$

so that $\left\|[F_0 \cdot c_0]_q\right\|_\infty < nE^2$ and $[F_0 \cdot c_0]_q \pmod 2 = m \cdot m'$. Furthermore, for all $j \in [t]$,

$$\begin{aligned}[F_j \cdot c_j]_q &= \left[F_j \cdot \left\langle \mathsf{Bit}\left(c_{j-1}\right), h_{i_j}\widetilde{\mathbf{s}} + 2\widetilde{\mathbf{e}} + \mathsf{Pow}\left(f_{i_j}\right)\right\rangle\right]_q \\ &= \left[F_j \cdot \left\langle \mathsf{Bit}\left(c_{j-1}\right), h_{i_j}\widetilde{\mathbf{s}}\right\rangle + F_j \cdot \left\langle \mathsf{Bit}\left(c_{j-1}\right), 2\widetilde{\mathbf{e}}\right\rangle + F_j c_{j-1} f_{i_j}\right]_q \\ &= F_j f_{i_j}^{-1} \cdot \left\langle \mathsf{Bit}\left(c_{j-1}\right), 2g_{i_j}\widetilde{\mathbf{s}}\right\rangle + F_j \cdot \left\langle \mathsf{Bit}\left(c_{j-1}\right), 2\widetilde{\mathbf{e}}\right\rangle + F_{j-1} c_{j-1}\end{aligned}$$

Using the fact that each F_j is the product of at most $(2N - j)$ keys, we have that

$$\begin{aligned}\left\|[F_j \cdot c_j]_q\right\|_\infty &< 2\left\lceil \log q \right\rceil n^2 B^2 \cdot (nB)^{2N-j-1} + 2\left\lceil \log q \right\rceil nB \cdot (nB)^{2N-j} + \left\|[F_{j-1} \cdot c_{j-1}]_q\right\|_\infty \\ &= 4\left\lceil \log q \right\rceil (nB)^{2N-j+1} + \left\|[F_{j-1} \cdot c_{j-1}]_q\right\|_\infty\end{aligned}$$

This yields the following bound on the error of c_{mult}:

$$\begin{aligned}\left\|[F_{K \cup K'} \cdot c_{\mathsf{mult}}]_q\right\|_\infty = \left\|[F_t \cdot c_t]_q\right\|_\infty &\leq nE^2 + \sum_{j=1}^{t} 4\left\lceil \log q \right\rceil (nB)^{2N-j+1} \\ &= nE^2 + 4\left\lceil \log q \right\rceil (nB)^{2N+1} \sum_{j=1}^{t} (nB)^{-j} \\ &\leq nE^2 + 8\left\lceil \log q \right\rceil (nB)^{2N+1} \\ &\leq (nB)^{2N} E^2\end{aligned}$$

where the last inequality holds by the fact that $q = 2^{n^\varepsilon}$.

Furthermore, notice that $[F_j \cdot c_j]_q \equiv F_{j-1} c_{j-1} \pmod 2$. Since $[F_0 \cdot c_0]_q \pmod 2 = m \cdot m'$, we can conclude that $[F_{K \cup K'} \cdot c_{\mathsf{mult}}]_q \pmod 2 = [F_t \cdot c_t]_q \pmod 2 = m \cdot m'$.

□

Once we have bounded the magnitude of the error coefficients after a homomorphic operation, we can bound the overall error incurred after evaluating a circuit of depth D. Starting with error $E_0 \leq 3(nB)^2$, after D levels of homomorphic operations, the error magnitude can grow to at most:

$$\left((nB)^{2N} E_0\right)^{2^D} \leq \left((3nB)^{2^D \cdot (2N+2)}\right)$$

This results in correct decryption as long as $D < \log\log q - \log\log n - \log N - O(1)$, where we use the fact that $B = \mathsf{poly}(n)$. In particular, for $N = O(n^\delta)$ keys and $q = 2^{n^\varepsilon}$, we get $D < (\varepsilon - \delta)\log n - \log\log n - O(1)$.

□

5.3.3 Security

Recall from Section 2.8 that the security of the (modified) NTRU encryption scheme can be based on two assumptions – the RLWE assumption and the DSPR assumption. Recall further that Stehlé and Steinfeld [SS11b] showed that the $\mathsf{DSPR}_{\phi,q,\chi}$ assumption is unconditionally true if n is a power of 2, $\phi(x) = x^n + 1$ is the n^{th} cyclotomic polynomial, and χ is the discrete Gaussian $D_{\mathbb{Z}^n,r}$ for $r > \sqrt{q} \cdot \mathsf{poly}(n)$. This allowed them to prove semantic security for the modified NTRU scheme under the $\mathsf{RLWE}_{\phi,q,\chi}$ assumption alone.

Unfortunately, their result holds only if $r > \sqrt{q} \cdot \mathsf{poly}(n)$, which is too large to permit even a single homomorphic multiplication. To support homomorphic operations, we need to use a much smaller value of r, for which it is easy to see that the $\mathsf{DSPR}_{\phi,q,\chi}$ assumption does not hold in a statistical sense any more. Thus, it is necessary to assume that the decisional small polynomial ratio problem is hard for our choice of parameters.

Additionally, note that the evaluation key ek contains elements of the form $[hs_\tau + 2e_\tau + 2^\tau f]_q$, which can be thought of as "pseudo-encryptions" of (multiples of) the secret key f under the corresponding public key h.[5] The security of the scheme must then additionally rely on a "circular security" assumption that states that semantic security of the scheme is maintained given the evaluation key as constructed above. We remark that this assumption can be avoided at the cost of obtaining a leveled homomorphic encryption scheme (where the size of the evaluation key grows with the depth of the circuits that the scheme supports).

Thus, we can base the security of the scheme on the DSPR assumption, the RLWE assumption, and the "circular security" assumption described above.

Lemma 5.3.2. *Let n be a power of 2, let $\phi(x) = x^n + 1$, let $q = 2^{n^\varepsilon}$ for $\varepsilon \in (0,1)$ and $\chi = D_{\mathbb{Z}^n,r}$ with $r = \mathsf{poly}(n)$. Then the somewhat homomorphic encryption scheme $\mathcal{E}_{\mathsf{SH}} = \{\mathsf{SH.Keygen}, \mathsf{SH.Enc}, \mathsf{SH.Dec}, \mathsf{SH.Eval}$ described above is secure under the $\mathsf{DSPR}_{\phi,q,\chi}$ and $\mathsf{RLWE}_{\phi,q,\chi}$ assumptions, and the assumption that the scheme remains semantically secure even given the evaluation key ek.*

5.4 From Somewhat to Fully Homomorphic Encryption

We use a generalization of Gentry's bootstrapping theorem [Gen09b, Gen09a] (see Section 2.5) to convert the multikey somewhat homomorphic scheme from Section 5.3 into a multikey fully homomorphic one. We modify Gentry's bootstrapping theorem and the corresponding definitions from their original presentation to generalize them to the multikey setting.

[5] We point out that these are not true encryptions of the "message" $2^\tau f$ since $2^\tau f$ is not a binary polynomial, whereas our scheme is only equipped to correctly encrypt polynomials $m \in R_2$.

Definition 5.4.1 (Multikey Bootstrappable Schemes)**.** *Let* $\mathcal{E} = \left\{\mathcal{E}^{(N)} = (\mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})\right\}_{N>0}$ *be a family of multikey* $\mathcal{C}$*-homomorphic encryption schemes, and let* f_{add} *and* f_{mult} *be the the augmented decryption functions of the scheme defined as*

$$f_{\mathsf{add}}^{c_1,c_2}(\mathsf{sk}_1, \ldots, \mathsf{sk}_N) = \mathsf{Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_N, c_1) \quad XOR \quad \mathsf{Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_N, c_2)$$

$$f_{\mathsf{mult}}^{c_1,c_2}(\mathsf{sk}_1, \ldots, \mathsf{sk}_N) = \mathsf{Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_N, c_1) \quad AND \quad \mathsf{Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_N, c_2)$$

Then $\mathcal{E}$ *is* bootstrappable *if* $\left\{f_{\mathsf{add}}^{c_1,c_2}, f_{\mathsf{mult}}^{c_1,c_2}\right\}_{c_1,c_2} \subseteq \mathcal{C}$*. Namely, the scheme can homomorphically evaluate* f_{add} *and* f_{mult}*.*

We now state a generalization of Gentry's bootstrapping theorem to the multikey setting. Taking $N = 1$ yields the theorem and the definitions from [Gen09b, Gen09a] and Section 2.5.

Theorem 5.4.1 (Multikey Bootstrapping Theorem)**.** *Let* $\mathcal{E}$ *be a bootstrappable family of multikey homomorphic schemes that is also weakly circular secure. Then there is a multikey fully homomorphic family of encryption schemes* $\mathcal{E}'$*.*

Unfortunately, the somewhat homomorphic scheme described in Section 5.3 is not bootstrappable. Recall that the scheme can only evaluate circuits of depth less than $\varepsilon \log(n)$, where $\varepsilon < 1$. However, the shallowest implementation of the decryption circuit we are aware of has depth $c \log N \cdot \log n$ for some constant $c > 1$.We therefore turn to modulus reduction, a technique introduced by [BV11a] and refined by [BGV12], to convert our somewhat homomorphic scheme into a bootstrappable scheme.

5.4.1 Modulus Reduction

Modulus reduction [BV11a, BGV12] is a noise-management technique that provides an *exponential* gain on the depth of the circuits that can be evaluated, while keeping the depth of the decryption circuit unchanged. Informally, if D_{dec} is the depth of the decryption circuit of the multikey scheme described in Section 5.3.1, then we modify the scheme so that its decryption circuit is unchanged but the scheme can now evaluate circuits of depth D_{dec}. Hence, the new scheme can evaluate its own decryption circuit, making it bootstrappable. Details follow.

Let (h, f) be a key pair and let c be a ciphertext under public key h. Recall that for decryption to be successful, we need the error $[fc]_q$ to be equal to $fc \in R$. However, each homomorphic operation increases this error. Modulus reduction allows us to keep the error magnitude small by simply scaling the ciphertext after each operation. The key idea is to exploit the difference in how the error affects security and homomorphism:

- The growth of error upon homomorphic multiplication is governed by the *magnitude* of the noise.
- Security is governed by the *ratio* between the magnitude of the error and the modulus q.

This suggests a method of reducing the magnitude of the error and the modulus by roughly the same factor, thus preserving security while at the same time making homomorphic multiplications "easier". In particular, modulus reduction gives us a way to transform a ciphertext $c \in R_q$ into a different ciphertext $c' \in R_p$ (for $p < q$) while preserving correctness: for "joint" secret key $f = \prod_{i=1}^{N} f_i$,

$$[fc]_p = [fc']_q \pmod 2$$

The transformation from c to c' involves simply scaling by (p/q) and rounding appropriately.

Lemma 5.4.2 ([BGV12])**.** *Let p and q be two odd moduli, and let $c \in R_q$. Define c' to be the polynomial in R_p closest to $(p/q) \cdot c$ such that $c' \equiv c \pmod 2$. Then, for any f with $\|[fc]_q\|_\infty < q/2 - (q/p) \cdot \|f\|_1$, we have*

$$[fc']_p = [fc]_q \pmod 2 \qquad \text{and} \qquad \left\|[fc']_p\right\|_\infty < (p/q) \cdot \|[fc]_q\|_\infty + \|f\|_1$$

where $\|\cdot\|_\infty$ and $\|\cdot\|_1$ are the ℓ_∞ and ℓ_1, respectively.

Proof. Let $fc = \sum_{i=0}^{n-1} d_i x^i$, and consider a coefficient d_i. We know that there exists $k \in \mathbb{Z}$ such that:

$$[d_i]_q = d_i - kq \in \left[-\frac{q}{2} + \frac{q}{p}\|f\|_1, \frac{q}{2} - \frac{q}{p}\|f\|_1\right],$$

so that

$$(p/q) \cdot d_i - kp \in \left[-\frac{p}{2} + \|f\|_1, \frac{p}{2} - \|f\|_1\right]$$

Let $fc' = \sum_{i=0}^{n-1} e_i x^i$. Then $-\|f\|_1 \le (p/q) \cdot e_i - d_i \le \|f\|_1$. Therefore,

$$e_i - kp \in \left[-\frac{p}{2}, \frac{p}{2}\right] \qquad \text{and} \qquad [e_i]_p = e_i - kp$$

This proves the second part of the lemma. To prove the first part, notice that since p and q are both odd, we know $kp \equiv kq \pmod 2$. Moreover, we chose c' such that $c \equiv c' \pmod 2$. We thus have

$$\begin{aligned} e_i - kp &\equiv d_i - kq \pmod 2 \\ [e_i]_p &\equiv [d_i]_q \pmod 2 \\ [fc']_p &\equiv [fc]_q \pmod 2 \end{aligned}$$

□

The beauty of Lemma 5.4.2 is that if we know the depth D of the circuit we want to evaluate, then we can construct a ladder of decreasing moduli $q_0, \ldots, q_D$ and perform modulus reduction after each operation so that at level i all ciphertexts reside in R_{q_i} and the magnitude of the noise at each level is small. In particular, this is true at level D so that once the evaluation is complete, it is possible to decrypt the resulting ciphertext without decryption errors. This yields a *leveled homomorphic encryption scheme*. A bootstrappable scheme can then be obtained by setting $D = D_{\mathsf{dec}}$, the depth of the augmented decryption circuit.

5.4.2 Obtaining A Leveled Homomorphic Scheme

We change the somewhat homomorphic scheme from Section 5.3 to use modulus reduction during the evaluation. The main changes to the scheme are as follows:

- The scheme is now additionally parametrized by an integer D, which is the maximum circuit depth that it can homomorphically evaluate, and a ladder of decreasing moduli $q_0, \ldots q_D$.
- We cannot use a single key f for all levels (at the expense of assuming the circular security), as in Section 5.3. This is because the public key h depends on the modulus q (recall that $h = 2gf^{-1}$, where f^{-1} is the inverse of f in R_q). With the new ladder of moduli, this

would require that the following two conditions be met simultaneously: First, that f^{-1} is the inverse of f in R_{q_D} (to guarantee correctness of decryption) and second, that $h = 2gf^{-1}$ is (indistinguishable from) uniformly random in R_{q_0} (to guarantee semantic security). This would require making a much stronger (and perhaps false) hardness assumption.

Instead, key generation computes a different key pair $\left(h^{(d)}, f^{(d)}\right)$ for each level $d \in \{0, \ldots, D\}$. Encryption uses $\mathsf{pk} \stackrel{\mathsf{def}}{=} h^{(0)}$ as the public key, and decryption uses $\mathsf{sk}^{(d)} \stackrel{\mathsf{def}}{=} f^{(d)}$ to decrypt a "level-d" ciphertext, ie. a ciphertext that is the output of a depth-d circuit evaluation. W.l.o.g. we assume any ciphertext to be decrypted is a level-D ciphertext and set the secret key to be $\mathsf{sk} = f^{(D)}$.

Homomorphic operations will ensure that if c, c' are level-$(d-1)$ ciphertexts in $R_{q_{d-1}}$ decryptable with $f^{(d-1)}$, then c_{add} and c_{mult} are level-d ciphertexts in R_{q_d} decryptable with $f^{(d)}$.

- Relinearization will now serve two purposes: it will ensure that only linear terms of keys are needed to decrypt the resulting ciphertext, but it will also switch the level-$(d-1)$ key to the corresponding level-(d) key. (Indeed, relinearization is also known as *key-switching* in the literature). Moreover, note that we must perform the key-switching step not only for quadratic terms but also for linear terms. Thus, we now perform relinearization/key-switching after *every* homomorphic operation, both addition and multiplication, and furthermore, we relinearize/key-switch every key in $K \cup K'$, instead of only those in $K \cap K'$.

- To perform the relinearization/key-switching step described above, the evaluation key consists of pseudo-encryptions of $f^{(d-1)}$ and $\left(f^{(d-1)}\right)^2$ under the public key $h^{(d)}$, for all $d \in [D]$.

  Note in particular that we now need pseudo-encryptions of the quadratic terms of the key. In the scheme from Section 5.3, relinearization only required pseudo-encryptions of (multiples of) f because the term $\langle \mathsf{Bit}(c), \mathsf{Pow}(f) \rangle$ only performed "partial decryption" of the ciphertext c; it computes fc but f^2 is required to decrypt c. Decryption of c was completed at decryption time when the ciphertext was multiplied by f once more, obtaining $f^2 c$.

  In our new setting, because decryption is performed using a *different* key, relinearization needs to "completely decrypt" c with the original key. For a key in $K \cap K'$, this means computing $\left[\left\langle \mathsf{Bit}(c), \mathsf{Pow}\left(\left(f^{(d-1)}\right)^2\right)\right\rangle\right]_q = \left[\left(f^{(d-1)}\right)^2 c\right]_q$. Since $\mathsf{Pow}\left(\left(f^{(d-1)}\right)^2\right)$ is encrypted under $h^{(d)}$, the new ciphertext can be decrypted using $f^{(d)}$.

  Pseudo-encryptions of the linear terms of the keys are also required in order to relinearize/key-switch keys in $K \triangle K'$, the symmetric difference of K, K'.

Formal Description

We now give a formal description of the leveled homomorphic encryption scheme resulting from applying the changes described above to the somewhat homomorphic scheme $\mathcal{E}_{\mathsf{SH}}$ described in Section 5.3.

- $\mathsf{LH.Keygen}(1^\kappa)$: For every $i \in \{0, \ldots, D\}$, sample $g^{(i)}, u^{(i)} \leftarrow \chi$ and set $f^{(i)} := 2u^{(i)} + 1$ so that $f^{(i)} \equiv 1 \pmod 2$. If $f^{(i)}$ is not invertible in R_{q_i}, resample $u^{(i)}$; otherwise, let $\left(f^{(i)}\right)^{-1}$

be the inverse of $f^{(i)}$ in R_q. Let $h^{(i)} \stackrel{\mathsf{def}}{=} \left[2g^{(i)}\left(f^{(i)}\right)^{-1}\right]_{q_i} \in R_{q_i}$, and set

$$\mathsf{pk} \stackrel{\mathsf{def}}{=} h^{(0)} \in R_{q_0} \quad , \quad \mathsf{sk} \stackrel{\mathsf{def}}{=} f^{(D)} \in R_{q_D}$$

For all $i \in [D]$, sample $\widetilde{\mathbf{s}}_{\gamma}^{(i)}, \widetilde{\mathbf{e}}_{\gamma}^{(i)}, \widetilde{\mathbf{s}}_{\zeta}^{(i)}, \widetilde{\mathbf{e}}_{\zeta}^{(i)} \leftarrow \chi^{\lceil \log q \rceil}$ and compute

$$\boldsymbol{\gamma}^{(i)} := \left[h^{(i)}\widetilde{\mathbf{s}}_{\gamma}^{(i)} + 2\widetilde{\mathbf{e}}_{\gamma}^{(i)} + \mathsf{Pow}\left(f^{(i-1)}\right)\right]_{q_i} \in R_{q_i}^{\lceil \log q_i \rceil}$$
$$\boldsymbol{\zeta}^{(i)} := \left[h^{(i)}\widetilde{\mathbf{s}}_{\zeta}^{(i)} + 2\widetilde{\mathbf{e}}_{\zeta}^{(i)} + \mathsf{Pow}\left(\left(f^{(i-1)}\right)^2\right)\right]_{q_i} \in R_{q_i}^{\lceil \log q_i \rceil}$$

Set $\mathsf{ek} \stackrel{\mathsf{def}}{=} \left\{\boldsymbol{\gamma}^{(i)}, \boldsymbol{\zeta}^{(i)}\right\}_{i \in [D]}$, and output the key tuple $(\mathsf{pk}, \mathsf{sk}, \mathsf{ek})$.

- $\mathsf{LH.Enc}(\mathsf{pk}, m)$: Sample $s, e \leftarrow \chi$. Output the ciphertext $c := [hs + 2e + m]_{q_0} \in R_{q_0}$.
- $\mathsf{LH.Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_N, c)$: Assume w.l.o.g. that $c \in R_{q_D}$. Parse $\mathsf{sk}_i = f_i$ for $i \in [N]$. Let $\mu := [f_1 \cdots f_N \cdot c]_{q_D} \in R_{q_D}$. Output $m' := \mu \pmod 2$.
- $\mathsf{LH.Eval}(C, (c_1, \mathsf{pk}_1, \mathsf{ek}_1), \ldots, (c_\ell, \mathsf{pk}_\ell, \mathsf{ek}_\ell))$: We show how to evaluate an ℓ-variate boolean circuit $C : \{0,1\}^\ell \to \{0,1\}$ of depth D. To this end, we show how to homomorphically add and multiply two elements in $\{0,1\}$. As before, given two ciphertexts c, c', we assume that we also have a set of distinct public keys associated with each ciphertext, and denote these lists by K, K', respectively. The public-key set of a fresh encryption is simply the set $\{\mathsf{pk}\}$ containing the public key under which it was encrypted. For convenience, in our analysis we sometimes assume that the set contains the indices of the public keys instead of the keys themselves.

  - Given two ciphertexts $c, c \in R_{q_d}$ with corresponding public-key sets K, K', compute $c_0 = [c + c']_{q_d} \in R_{q_d}$ and let $K \cup K' = \left\{\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_t}\right\}$. For $j = 1, \ldots, r$, parse $\mathsf{ek}_{i_j} = \left\{\boldsymbol{\gamma}_{i_j}^{(\delta)}, \boldsymbol{\zeta}_{i_j}^{(\delta)}\right\}_{\delta \in [D]}$ and compute

    $$c_j = \left[\left\langle \mathsf{Bit}\left(c_{j-1}\right), \boldsymbol{\gamma}_{i_j}^{(d)} \right\rangle\right]_q \in R_{q_d}$$

    Finally, reduce the modulus: let c_{add} be the integer vector closest to $(q_{d+1}/q_d) \cdot c_t$ such that $c_{\mathsf{add}} \equiv c_t \pmod 2$. Output $c_{\mathsf{add}} \in R_{q_{d+1}}$ as an encryption of the *sum* of the underlying messages. Output the set $K_{\mathsf{add}} \stackrel{\mathsf{def}}{=} K \cup K'$ as its corresponding public-key set.
  - Given two ciphertexts $c, c \in R_{q_d}$ with corresponding public-key sets K, K', compute $c_0 = [c + c']_{q_d} \in R_{q_d}$ and let $K \cup K' = \left\{\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_t}\right\}$. For $j = 1, \ldots, r$, parse $\mathsf{ek}_{i_j} = \left\{\boldsymbol{\gamma}_{i_j}^{(\delta)}, \boldsymbol{\zeta}_{i_j}^{(\delta)}\right\}_{\delta \in [D]}$ and compute c_j as follows:
    - If $\mathsf{pk}_{i_j} \in K \cap K'$, let

      $$c_j = \left[\left\langle \mathsf{Bit}\left(c_{j-1}\right), \boldsymbol{\gamma}_{i_j}^{(d)} \right\rangle\right]_q \in R_{q_d}$$

∗ Otherwise, let

$$c_j = \left[\left\langle \mathsf{Bit}\left(c_{j-1}\right), \boldsymbol{\zeta}_{i_j}^{(d)} \right\rangle\right]_q \in R_{q_d}$$

Finally, reduce the modulus: let c_{mult} be the integer vector closest to $(q_{d+1}/q_d) \cdot c_t$ such that $c_{\mathsf{mult}} \equiv c_t \pmod 2$. Output $c_{\mathsf{mult}} \in R_{q_{d+1}}$ as an encryption of the *product* of the underlying messages. Output the set $K_{\mathsf{mult}} \stackrel{\mathsf{def}}{=} K \cup K'$ as its corresponding public-key set.

Leveled Homomorphism. We can now show the following lemma, characterizing the circuits and number of keys that the scheme can handle in evaluation.

Lemma 5.4.3. *Let χ is a B-bounded distribution for $B = \mathsf{poly}(n)$, let $q_0 = 2^{n^\varepsilon}$ for $\varepsilon \in (0,1)$ and for $d \in [D]$, let $q_{d-1}/q_d = 8n(nB)^{2N+2}$. Then the encryption scheme $\mathcal{E}_{\mathsf{LH}} = (\mathsf{LH.Keygen}, \mathsf{LH.Enc}, \mathsf{LH.Dec}, \mathsf{LH.Eval})$ described above is multikey homomorphic for N keys and circuits of depth D as long as $ND = O\left(n^\varepsilon / \log n\right)$.*

Proof. We claim that for all $d \in \{0, \ldots, D\}$, the error of a level-d ciphertext is bounded by $E \stackrel{\mathsf{def}}{=} (1/2n) \cdot (q_{d-1}/q_d) = 4(nB)^{2N+2}$, and prove it by induction. The base case follows immediately since the error of a freshly encrypted ciphertext is bounded by $3(nB)^2 < 4(nB)^{2N+2}$.

We now turn to the inductive step. Let c, c' be level-$(d-1)$ ciphertexts with corresponding public key sets K, K'. The inductive hypothesis tells us the error in c and c' is bounded by E. Using the same analysis as in the proof of Lemma 5.3.1, we can show that relinearizing all keys in $K \cup K'$ generates an additive error less than $8\left\lceil \log q_d \right\rceil (nB)^{2N+1} < (nB)^{2N+2}$, where we used the fact that $q_d < q_0 = 2^{n^\varepsilon}$ for $\varepsilon < 1$. Recall that c_t is the ciphertext obtained in a homomorphic operation after relinearization has been completed but before modulus reduction is performed. Then:

- In a homomorphic addition, the error of c_t is bounded by $2(nB)^N E + (nB)^{2N+2}$. By Lemma 5.4.2, the error of c_{add} is bounded by:

$$\begin{aligned}
\frac{q_d}{q_{d-1}} \cdot \left(2(nB)^N E + (nB)^{2N+2}\right) + \|f\|_1 &\leq \frac{2(nB)^N E + (nB)^{2N+2}}{2nE} + nB \\
&\leq \frac{2(nB)^N E}{2nE} + (nB)^{2N+2} + nB \\
&\leq \frac{(nB)^N}{n} + (nB)^{2N+2} + nB \\
&\leq 4(nB)^{2N+2} = E
\end{aligned}$$

- In a homomorphic multiplication, the error of c_t is bounded by $nE^2 + (nB)^{2N+2}$. By Lemma 5.4.2, the error of c_{mult} is bounded by:

$$\begin{aligned}
\frac{q_d}{q_{d-1}} \cdot \left(nE^2 + (nB)^{2N+2}\right) + \|f\|_1 &\leq \frac{nE^2 + (nB)^{2N+2}}{2nE} + nB \\
&\leq \frac{nE^2}{2nE} + 2(nB)^{2N+2} \\
&\leq \frac{E}{2} + \frac{E}{2} = E
\end{aligned}$$

This concludes the inductive step and the proof that ciphertexts of all levels have error at most E.

To correctly decrypt a level-D ciphertext, we must have that

$$(nB)^{2N+2} = E < \frac{q_D}{2} < \frac{q_0}{2(8n(nB)^{2N+2})^D} = \frac{2^{n^\varepsilon}}{2(8n(nB)^{2N+2})^D}$$

which yields the theorem statement: $ND = O\left(n^\varepsilon / \log n\right)$. □

Security. As in Section 5.3, the security of the scheme $\mathcal{E}_{\mathsf{LH}}$ can be based in the $\mathsf{DSPR}_{\phi,q,\chi}$ and $\mathsf{RLWE}_{\phi,q,\chi}$ assumptions. However, unlike in Section 5.3, we do not need to assume circular security of the encryption scheme. This is due to the fact that the evaluation key consists of pseudo-encryptions of (multiples of) $f^{(d-1)}$ and $\left(f^{(d-1)}\right)^2$ under a *different* public key $h^{(d)}$, for all $d \in [D]$. Semantic security even given the evaluation key can then be established by a hybrid argument that converts all pseudo-encryptions in the evaluation key, one-by-one, to uniform elements in R_q.

Lemma 5.4.4. *Let n be a power of 2, let $\phi(x) = x^n + 1$, let $q = 2^{n^\varepsilon}$ for $\varepsilon \in (0,1)$ and $\chi = D_{\mathbb{Z}^n,r}$ with $r = \mathsf{poly}(n)$. Then the multikey leveled homomorphic encryption scheme $\mathcal{E}_{\mathsf{LH}} = (\mathsf{LH.Keygen}, \mathsf{LH.Enc}, \mathsf{LH.Dec}, \mathsf{LH.Eval})$ described above is secure under the $\mathsf{DSPR}_{\phi,q,\chi}$ and $\mathsf{RLWE}_{\phi,q,\chi}$ assumptions.*

5.4.3 Multikey Fully Homomorphic Encryption

To convert the leveled homomorphic encryption scheme described in Section 5.4.2 into a fully homomorphic scheme, we use the multikey bootstrapping theorem (Theorem 5.4.1). First, we show an upper bound on the depth of the decryption circuit and show that the scheme is bootstrappable.

Lemma 5.4.5. *The N-key decryption circuit of the leveled homomorphic encryption scheme described above can be implemented as a polynomial-size arithmetic circuit over $GF(2)$ of depth $O(\log N \cdot (\log\log q_D + \log n))$.*

Proof. The decryption circuit for a ciphertext encrypted under N keys can be written as

$$\mathsf{Dec}(f_1, \ldots, f_N, c) = c \cdot \prod_{i=1}^{N} f_i$$

Multiplying two polynomials over R_{q_D} can be done using a polynomial-size Boolean circuit of depth $O(\log\log q_D + \log n)$ (see, e.g., [BV11a, Lemma 4.5] for a proof). Using a binary tree of polynomial multiplications, the decryption operation above can then be done in depth $O(\log N \cdot (\log\log q_D + \log n))$, as claimed. □

This means that the modified scheme is bootstrappable, and therefore applying the bootstrapping theorem gives us:

Theorem 5.4.6. *Let χ is a B-bounded distribution for $B = \mathsf{poly}(n)$, let $q_0 = 2^{n^\varepsilon}$ for $\varepsilon \in (0,1)$ and for $d \in [D]$, let $q_{d-1}/q_d = 8n(nB)^{2N+2}$. Then, there exists a multikey fully homomorphic encryption scheme for $N = O(n^\varepsilon / \log^3 n)$ keys, secure under the $\mathsf{DSPR}_{\phi,q,\chi}$ and $\mathsf{RLWE}_{\phi,q,\chi}$ assumptions, and the assumption that the leveled homomorphic encryption scheme $\mathcal{E}_{\mathsf{LH}} = (\mathsf{LH.Keygen}, \mathsf{LH.Enc}, \mathsf{LH.Dec}, \mathsf{LH.Eval})$ described above is weakly circular secure.*

Proof. To apply the multikey bootstrapping theorem (Theorem 5.4.1), we require that the depth of the decryption circuit is smaller than the depth of the circuits that the scheme can evaluate. That is, we require that

$$\log N \cdot (\log \log q_D + \log n) < C \cdot \frac{\log q_0}{N \cdot \log n}$$

for some universal constant $C > 0$. For $N \leq \sqrt{C/2} \cdot (n^{\varepsilon/2}/\log n)$, we have,

$$\begin{aligned} N \cdot \log n \cdot \log N \cdot (\log \log q_D + \log n) &\leq N^2 \cdot \log n \cdot \cdot (\log \log q_0 + \log n) \\ &\leq \frac{C}{2} \cdot \frac{n^{\varepsilon}}{\log^2 n} \cdot (1+\varepsilon) \cdot \log^2 n \\ &\leq C \cdot n^{\varepsilon} = C \cdot \log q_0 \end{aligned}$$

as required. □

Remark 5.4.2. *Theorem 5.4.6 implies that for any $N \in \mathbb{N}$, there exists a multikey fully homomorphic encryption scheme for N keys. This is achieved by choosing ε' such that $n^{\varepsilon'} \leq \sqrt{C/2} \cdot (n^{\varepsilon/2}/\log n)$ and setting $n \geq N^{1/\varepsilon'}$.*

We emphasize the fact that bootstrapping (and therefore assuming weak circular security) can be avoided at the cost of obtaining a leveled homomorphic encryption scheme.

Chapter 6
On-the-Fly MPC from Multikey FHE

We now show how to construct on-the-fly MPC from multikey FHE. We first construct a basic protocol that is secure against semi-malicious adversaries, and then describe how to modify the protocol to obtain security against malicious adversaries. As mentioned earlier, the main building block of our construction is multikey fully homomorphic encryption, defined and constructed in Chapter 5.

6.1 The Basic Protocol

Let $\left\{\mathcal{E}^{(N)} = (\mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})\right\}_{N>0}$ be a multikey fully-homomorphic family of encryption schemes. The following construction is an on-the-fly MPC protocol secure against semi-malicious adversaries. The protocol is defined as follows:

Step 1: For $i \in [U]$, party P_i samples a key tuple $(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i) \leftarrow \mathsf{Keygen}(1^\kappa)$ and encrypts its input x_i under pk_i: $c_i \leftarrow \mathsf{Enc}(\mathsf{pk}_i, x_i)$. It sends $(\mathsf{pk}_i, \mathsf{ek}_i, c_i)$ to the server S.

At this point a function F, represented as a circuit C, has been selected on inputs $\{x_i\}_{i\in V}$ for some $V \subseteq U$. Let $N = |V|$. For ease of notation, assume w.l.o.g. that $V = [N]$. The parties proceed as follows.

Step 2: The server S computes $c := \mathsf{Eval}(C, (c_1, \mathsf{pk}_1, \mathsf{ek}_1), \ldots, (c_N, \mathsf{pk}_N, \mathsf{ek}_N))$ and broadcasts c to parties $P_1, \ldots, P_N$.

Step 3: The parties $P_1, \ldots, P_N$ run a secure MPC protocol $\Pi_{\text{DEC}}{}^{\text{SM}}$ to compute the function $g_c(\mathsf{sk}_1, \ldots, \mathsf{sk}_N) \stackrel{\mathsf{def}}{=} \mathsf{Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_N, c)$.

We remark that an upper bound on the number of computing parties must be known in advance. This is a direct consequence of the "leveled" nature of our multikey FHE construction with respect to the number of keys.

6.1.1 Security Against Semi-Malicious Adversaries

Theorem 6.1.1. *Let $\left\{\mathcal{E}^{(N)} = (\mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})\right\}_{N>0}$ be a multikey fully-homomorphic encryption scheme, and let $\Pi_{\text{DEC}}{}^{\text{SM}}$ be an N-party MPC protocol for computing the decryption function $g_c(\mathsf{sk}_1, \ldots, \mathsf{sk}_N) \stackrel{\mathsf{def}}{=} \mathsf{Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_N, c)$. If $\mathcal{E}$ is semantically secure, and $\Pi_{\text{DEC}}{}^{\text{SM}}$ is secure against semi-honest adversaries corrupting $t < N$ parties, then the above construction is an on-the-fly MPC protocol secure against (static) semi-malicious adversaries corrupting t parties and possibly the server S.*

Proof. We prove that the protocol is correct and secure, and that it satisfies the performance requirements of an *on-the-fly* protocol.

Correctness: Correctness follows directly from the correctness properties of homomorphic evaluation and the MPC protocol ${\Pi_{\text{DEC}}}^{\text{SM}}$ for decryption.

Performance: By compactness of evaluation, we know that c is independent of $|C|$. This means that the communication complexity and the computation time of the parties is independent of $|C|$.

Security: We show security for the case when the server is corrupted; the case when the server is honest is analogous. Let $\mathcal{A}^{\text{SM}}$ be a real-world semi-malicious adversary corrupting t clients and the server. Recall that for security, we only need to consider adversaries corrupting a subset T of the parties $P_1, \ldots, P_N$ involved in the computation. Thus, we assume $t < N$, let $T \subsetneq [N]$ be the set of corrupted clients, and let $\overline{T} = [N] \backslash T$.

We construct a simulator $\mathcal{S}^{\text{SM}}$ as follows. The simulator receives the inputs of the corrupted parties, $\{x_i\}_{i \in T}$ and runs $\mathcal{A}^{\text{SM}}$ on these inputs $\{x_i\}_{i \in T}$. It simulates the messages for all honest parties in the protocol execution with $\mathcal{A}^{\text{SM}}$ by sampling all key tuples correctly, but encrypting 0 instead of the honest input x_i (which it doesn't know). In Step 3, it runs the simulator $\mathcal{S}^{\text{SM}}_{{\Pi_{\text{DEC}}}^{\text{SM}}}$ for the protocol ${\Pi_{\text{DEC}}}^{\text{SM}}$.

Step 1: For non-computing parties $i \in \{N+1, \ldots, U\}$ and for honest parties $i \in \overline{T}$, $\mathcal{S}^{\text{SM}}$ computes $(\mathsf{pk}_i, \cdot, \mathsf{ek}_i) \leftarrow \mathsf{Keygen}(1^\kappa)$ honestly and computes $c_i \leftarrow \mathsf{Enc}(\mathsf{pk}_i, 0)$. For each party P_i, the simulator sends $(c_i, \mathsf{pk}_i, \mathsf{ek}_i)$ to $\mathcal{A}^{\text{SM}}$ on behalf of P_i.

At the end of this round, it reads from $\mathcal{A}^{\text{SM}}$'s witness tape the secret keys $\{\mathsf{sk}_i\}_{i \in T}$ and the inputs $\{\widetilde{x}_i\}_{i \in T}$. The simulator sends these inputs to the trusted functionality $\mathcal{F}$ and receives the output $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_N)$, where $\widetilde{x}_i = x_i$ for honest inputs $i \in \overline{T}$

Step 2: The simulator receives c from $\mathcal{A}^{\text{SM}}$ as the server's broadcast message.

Step 3: The simulator $\mathcal{S}^{\text{SM}}$ runs the simulator $\mathcal{S}^{\text{SM}}_{{\Pi_{\text{DEC}}}^{\text{SM}}}$ for the decryption protocol (interacting with $\mathcal{A}^{\text{SM}}$). When $\mathcal{S}^{\text{SM}}_{{\Pi_{\text{DEC}}}^{\text{SM}}}$ queries the ideal decryption functionality with secret keys $\left\{\widetilde{\mathsf{sk}}_i\right\}_{i \in T}$, $\mathcal{S}^{\text{SM}}$ returns $\widetilde{y}$.

Output: The simulator receives the output of the corrupted parties from $\mathcal{A}^{\text{SM}}$, and returns these as its output.

We prove that $\text{IDEAL}_{\mathcal{F}, \mathcal{S}^{\text{SM}}}(\vec{x}) \overset{c}{\approx} \text{REAL}_{\Pi^{\text{SM}}, \mathcal{A}^{\text{SM}}}(\vec{x})$ via a series of hybrids.

Hybrid 0: This is the real-world execution of the protocol.

Hybrid 1: We change how Step 3 is performed. Instead of executing the protocol ${\Pi_{\text{DEC}}}^{\text{SM}}$ where honest parties use their individual secret keys, we run the simulator $\mathcal{S}^{\text{SM}}_{{\Pi_{\text{DEC}}}^{\text{SM}}}$ (interacting with $\mathcal{A}^{\text{SM}}$). When $\mathcal{S}^{\text{SM}}_{{\Pi_{\text{DEC}}}^{\text{SM}}}$ queries the ideal decryption functionality with secret keys $\left\{\widetilde{\mathsf{sk}}_i\right\}_{i \in T}$, we return

$$\widetilde{y} = g_c(\widetilde{\mathsf{sk}}_1, \ldots, \widetilde{\mathsf{sk}}_N) = \mathsf{Dec}(\widetilde{\mathsf{sk}}_1, \ldots, \widetilde{\mathsf{sk}}_N, c)$$

where $\widetilde{\mathsf{sk}}_i = \mathsf{sk}_i$ for honest secret keys $i \in \overline{T}$. The output of the corrupted parties is defined to be the output of $\mathcal{S}^{\text{SM}}_{{\Pi_{\text{DEC}}}^{\text{SM}}}$, and the output of the honest parties is defined to be $\widetilde{y}$.

We claim that Hybrid 0 is computationally indistinguishable from Hybrid 1 by the *security of* ${\Pi_{\text{DEC}}}^{\text{SM}}$. Indeed, the security of the decryption protocol ${\Pi_{\text{DEC}}}^{\text{SM}}$ guarantees that as long as we correctly emulate the ideal decryption functionality, the joint output of all parties is computationally indistinguishable in a real-world execution of the protocol with adversary $\mathcal{A}^{\text{SM}}$ (Hybrid 0), and in an ideal-world execution of the protocol with adversary $\mathcal{S}^{\text{SM}}_{\Pi_{\text{DEC}}}$ (Hybrid 1). We correctly emulate the ideal decryption functionality, by definition.

Hybrid 2: We now change how we compute $\widetilde{y}$, the value returned to the simulator $\mathcal{S}^{\text{SM}}_{\Pi_{\text{DEC}}}$ when it queries the decryption ideal functionality. Instead of computing $\widetilde{y} = g_c(\widetilde{\mathsf{sk}}_1, \ldots, \widetilde{\mathsf{sk}}_N) = \mathsf{Dec}(\widetilde{\mathsf{sk}}_1, \ldots, \widetilde{\mathsf{sk}}_N, c)$, we instead compute

$$\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_N)$$

where $\widetilde{x}_i = x_i$ for honest inputs $i \in \overline{T}$, and where for corrupt parties $i \in T$, we recover $\widetilde{x}_i$ by reading $\mathcal{A}^{\text{SM}}$'s witness tape at the end of Step 1.

We claim that Hybrid 1 and Hybrid 2 are identically distributed. The adversary $\mathcal{A}^{\text{SM}}$ follows the protocol as specified, so in particular, it performs the homomorphic evaluation correctly. By *correctness of multikey evaluation* we know that c decrypts to $f(\widetilde{x}_1, \ldots, \widetilde{x}_N)$ when decrypted using the secret keys it computed in Step 1, $\{\mathsf{sk}_i\}_{i\in[N]}$; that is, $\mathsf{Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_N, c) = f(\widetilde{x}_1, \ldots, \widetilde{x}_N)$
Furthermore, because the adversary $\mathcal{A}^{\text{SM}}$ follows the protocol as specified, we know that the secret keys it uses in Step 3 are the same as the ones it computed in Step 1, i.e. $\mathsf{sk}_i = \widetilde{\mathsf{sk}}_i$ for all $i \in T$. We conclude that $\mathsf{Dec}(\widetilde{\mathsf{sk}}_1, \ldots, \widetilde{\mathsf{sk}}_N, c) = f(\widetilde{x}_1, \ldots, \widetilde{x}_N)$.

Hybrids $3.k$ for $k = 1, \ldots, N - t$: Let $\overline{T} = \{i_1, \ldots, i_{N-t}\}$. In Hybrid $3.k$ we change c_{i_k} so that instead of encrypting x_{i_k} it now encrypts 0. More formally, in Hybrid $3.k$ we have:

$$\left\{c_{i_j} \leftarrow \mathsf{Enc}(\mathsf{pk}_{i_j}, 0)\right\}_{j\le k} \quad , \quad \left\{c_{i_j} \leftarrow \mathsf{Enc}(\mathsf{pk}_{i_j}, x_{i_j})\right\}_{j>k}$$

For ease of notation we let Hybrid 2 be Hybrid 3.0. We claim that the view of $\mathcal{A}^{\text{SM}}$ in Hybrid $3.k$ is indistinguishable from its view in Hybrid $3.(k-1)$ by the *semantic security of* $\mathcal{E}$ under public key pk_{i_k}. Indeed, now that we run the simulator $\mathcal{S}^{\text{SM}}_{\Pi_{\text{DEC}}}$ in Step 3 instead of the real decryption protocol, the secret key sk_{i_k} is only used to encrypt c_{i_k}. So suppose, for the sake of contradiction, that there exists an algorithm $\mathcal{D}$ that distinguishes between hybrids $3.k$ and $3.(k-1)$. We construct an adversary $\mathcal{B}$ that breaks the semantic security of $\mathcal{E}$ under public key pk_{i_k}. The reduction $\mathcal{B}$ works as follows:

1. The reduction chooses arbitrary $\{x_i\}$.
2. It receives $(\mathsf{pk}, \mathsf{ek})$ from the semantic security challenger and sets $\mathsf{pk}_{i_k} = \mathsf{pk}$ and $\mathsf{ek}_{i_k} = \mathsf{ek}$. Gives $m_0 = 0$ and $m_1 = x_{i_k}$ to the challenger and receives $c = \mathsf{Enc}(\mathsf{pk}, m_b)$. Sets $c_{i_k} = c$. For all $i \in \overline{T}, i \neq i_k$, computes $(\mathsf{pk}_i, \cdot, \mathsf{ek}_i) \leftarrow \mathsf{Keygen}(1^\kappa)$ honestly. For $j < k$, computes $c_{i_j} \leftarrow \mathsf{Enc}(\mathsf{pk}_{i_j}, 0)$ and for $j > k$, computes $c_{i_j} \leftarrow \mathsf{Enc}(\mathsf{pk}_{i_j}, x_{i_j})$.
3. The reduction runs $\mathcal{A}^{\text{SM}}$: for all $i \in \overline{T}$ gives $(\mathsf{pk}_i, \mathsf{ek}_i, c_i)$ to $\mathcal{A}^{\text{SM}}$ on behalf of P_i, and receives c from $\mathcal{A}^{\text{SM}}$.

4. It reads from $\mathcal{A}^{\text{SM}}$'s witness tape the inputs $\{\widetilde{x}_i\}_{i\in T}$ and runs the simulator $\mathcal{S}^{\text{SM}}_{\Pi_{\text{DEC}}}$ (interacting with $\mathcal{A}^{\text{SM}}$). When $\mathcal{S}^{\text{SM}}_{\Pi_{\text{DEC}}}$ queries the ideal decryption functionality, it returns $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_N)$ where $\widetilde{x}_i = x_i$ for inputs $i \in \overline{T}$.
5. The reduction then gives $\mathcal{D}$ $\widetilde{y}$ as the output of all honest parties, as well as the output of $\mathcal{S}^{\text{SM}}_{\Pi_{\text{DEC}}}$.
6. Finally, $\mathcal{B}$ outputs the bit output by $\mathcal{D}$.

When $b = 0$, $\mathcal{B}$ perfectly emulates Hybrid $3.k$, whereas if $b = 1$, $\mathcal{B}$ perfectly emulates Hybrid $3.(k-1)$. Therefore, if $\mathcal{D}$ can distinguish between Hybrids $3.k$ and $3.(k-1)$, then $\mathcal{B}$ can distinguish between an encryption of m_0 and an encryption of m_1, contradicting the semantic security of $\mathcal{E}$.

We have proved that the joint output in Hybrid 0 is computationally indistinguishable from the joint output in Hybrid $3.(N-t)$. But notice that the joint output in Hybrid $3.(N-t)$ is precisely $\text{IDEAL}_{\mathcal{F},\mathcal{S}^{\text{SM}}}(\vec{x})$, and the joint output in Hybrid 0 is defined to be $\text{REAL}_{\Pi^{\text{SM}},\mathcal{A}^{\text{SM}}}(\vec{x})$. We conclude that $\text{IDEAL}_{\mathcal{F},\mathcal{S}^{\text{SM}}}(\vec{x}) \overset{c}{\approx} \text{REAL}_{\Pi^{\text{SM}},\mathcal{A}^{\text{SM}}}(\vec{x})$, as desired.

□

6.2 Achieving Security Against Malicious Adversaries

The protocol described in Section 6.1, though secure against semi-malicious adversaries, is not secure against fully malicious adversaries. We transform the protocol into one that is secure against malicious corruptions in three steps:

1. First, we replace the decryption protocol in Step 3 with one that is secure against malicious corruptions. More importantly, we change the function it computes to ensure that the secret key used in this protocol is consistent with the public and evaluation keys that the parties computed in Step 1.

2. Second, we add zero-knowledge proofs at each step in the protocol, following the AJW compiler [AJW11, AJL+12] (which is based on the GMW compiler [GMW87]).

3. Finally, in order to maintain the performance guarantees of the scheme, in Step 2 we replace the server's proof with a *succinct argument* (not necessarily ZK). This allows the server to prove that it correctly performed the homomorphic evaluation, and the clients to verify the validity of the proof in time that is significantly less than the size of the circuit.

6.2.1 The New Decryption Protocol

Our first step in handling malicious attacks is to replace the decryption protocol ${\Pi_{\text{DEC}}}^{\text{SM}}$ with one that is secure against malicious adversaries; we will denote it by ${\Pi_{\text{DEC}}}^{\text{MAL}}$. The function being computed by this protocol also needs to change in order to guarantee that the secret key used by each party is consistent with its public and evaluation keys:

$$g_{c,\mathsf{pk}_1,\mathsf{ek}_1,\ldots,\mathsf{pk}_N,\mathsf{ek}_N}((\mathsf{sk}_1, r_1)\ldots,(\mathsf{sk}_N, r_N))$$
$$\overset{\text{def}}{=} \begin{cases} \mathsf{Dec}(\mathsf{sk}_1,\ldots,\mathsf{sk}_N,c) & \text{if } (\mathsf{pk}_i,\mathsf{sk}_i,\mathsf{ek}_i) = \mathsf{Keygen}(1^\kappa; r_i) \ \ \forall i \in [N] \\ \bot & \text{otherwise} \end{cases}$$

Intuitively, if the protocol outputs something other than $\perp$, then in particular every corrupt party P_i "knows" a secret key $\widetilde{\mathsf{sk}}_i$ that is consistent with its public and evaluation keys $(\mathsf{pk}_i, \mathsf{ek}_i)$. By correctness of decryption, this binds P_i to the input $\widetilde{x}_i = \mathsf{Dec}\left(\widetilde{\mathsf{sk}}_i, \widetilde{c}_i\right)$, which by semantic security of the FHE, must be independent of the honest party's inputs.

We remark that the proceedings version of this work [LTV12] does not change the decryption function, but instead adds to Step 1 a zero-knowledge proof π_i^{GEN} for the relation $R^{\text{GEN}} = \{ \ (\ (\mathsf{pk}_i, \mathsf{ek}_i) \ , \ (\mathsf{sk}_i, r_i) \ ) \ \mid \ (\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i) := \mathsf{Keygen}(1^\kappa \ ; \ r_i) \ \}$. While this guarantees that the public and evaluation keys are well-formed, it does not guarantee that the secret key used in the decryption protocol in Step 3 is consistent with the public and evaluation keys $(\mathsf{pk}_i, \mathsf{ek}_i)$ created and used in Step 1. This allows a corrupt party to use a different secret key sk_i^* in Step 3 and potentially change the outcome of the decryption. We are therefore unable to prove security of that construction. However, the zero-knowledge proofs π_i^{GEN} can be required as an *optimization*, to guarantee that an honest server does not accept, store, or compute on ciphertexts that are encrypted under malformed keys (even though the outcome of any joint computation on such a ciphetext would not be decryptable using protocol ${\Pi_{\text{DEC}}}^{\text{MAL}}$).

Finally, we highlight the fact that if the protocol ${\Pi_{\text{DEC}}}^{\text{MAL}}$ can implemented using the cloud-assisted protocol described in Chapter 4. Jumping ahead, this yields a 5-round on-the-fly MPC protocol in the CRS-model, secure against malicious corruptions of any $t < [N]$ parties and possibly the server.

6.2.2 Adding Zero-Knowledge Proofs

The second step in our transformation is to apply the AJW compiler [AJW11, AJL$^+$12] (based on the GMW compiler [GMW87]) to the rest of the protocol (Steps 1 and 2), in order to ensure that parties do not deviate from the protocol specifications. This entails having each party and the server compute a zero-knowledge proof at every round, proving that their message in that round is well-formed and consistent with the protocol transcript.

Because the well-formedness of the public and evaluation keys $(\mathsf{pk}_i, \mathsf{ek}_i)$ is checked in the decryption protocol ${\Pi_{\text{DEC}}}^{\text{MAL}}$, the parties do not need to compute a separate zero-knowledge proof for this statement (unless required for the optimization described above). Therefore, each party only needs to prove that their ciphertext c_i is well-formed by providing a non-interactive zero-knowledge (NIZK) proof for the NP relation:

$$R^{\text{ENC}} = \{ \ (\ (\mathsf{pk}_i, c_i) \ , \ (x_i, s_i) \ ) \ \mid \ c_i = \mathsf{Enc}(\mathsf{pk}_i, x_i \ ; \ s_i) \ \}$$

We highlight the fact that the proof π_i^{ENC} must be *non-interactive*, for reasons that will become apparent shortly. Informally, this proof will either be broadcast by the server in Step 2 for all parties to verify, or it will be used as a witness in the proof of another NP relation. An interactive zero-knowledge proof would not be convincing it either of these cases since a valid proof transcript can always be simulated without knowing a witness and without the use of any trapdoors.

6.2.3 Maintaining Performance Guarantees.

Unfortunately, verifying a standard zero-knowledge proof for the server's computation in Step 2 requires time proportional to the size of the circuit. On the other hand, this computation is deterministic and public; indeed, anyone can verify the validity of the server's broadcast message by

performing the homomorphic evaluation themselves, albeit by also computing in time proportional to the size of the circuit. We solve this problem by replacing the server's proof with a *succinct argument* (not necessarily ZK), that allows the server to prove that it correctly performed the homomorphic evaluation, and the clients to verify the validity of the proof in time that is significantly less than the size of the circuit. We offer several solutions, each with its own benefits and drawbacks.

Verification for Small Inputs

We first consider the case where the ciphertexts $(c_1, \ldots, c_N)$ are small enough to be broadcast to the N parties in V, allowing communication complexity in the online phase to be linear in the total input size of the participating parties. In this case, the server will broadcast all ciphertexts and proofs $\{c_i, \pi_i^{\text{ENC}}\}_{i\in[N]}$, the evaluated ciphertext c, and a succinct argument φ showing that it performed the homomorphic evaluation correctly. The server needs to convince the participating parties that "$c = \mathsf{Eval}(C, (c_1, \mathsf{pk}_1, \mathsf{ek}_1), \ldots, (c_N, \mathsf{pk}_N, \mathsf{ek}_N))$", i.e., that a deterministic circuit of size $\mathsf{poly}(|C|, \kappa)$ accepts. For any uniform circuit C (i.e., computable by a $\mathsf{poly}(\kappa)$-time Turing machine), the following offer $\mathsf{poly}(\kappa, \log(|C|))$ communcation and verification efficiency.[1]

1. Use the argument system of Kilian [Kil92, Kil95], yielding *interactive* 4-round verification. It relies on expensive PCPs.

2. Use the succinct non-interactive arguments (SNARGs and SNARKs) of Micali [Mic94], Bitansky et al. [BCCT12, BCCT13] or Goldwasser et al. [GLR11] (see Section 2.3). These are non-interactive[2] but are secure only in the random oracle model [BR93] (in the case of CS proofs) or hold in the standard model but require a non-falsifiable assumption [Nao03]. Some variants rely on PCPs, PIR or FHE.

In case that the evaluation circuit is in logspace-uniform **NC**, we have another alternative:

4. Use the argument system of Goldwasser et al. [GKR08] for a 1-round solution[3]. It relies on PIR.

Unfortunately, we are unable to use verifiable computation protocols in the pre-processing model (e.g. [GGP10, CKV10, AIK10]) or SNARGs/SNARKs where the CRS depends on the circuit to be computed or where its size is at least as big as the computation, e.g. [Gro10, Lip12, GGPR13, PHGR13, Lip13]. These require the clients to participate in a pre-processing phase where their computation is proportional to the size of the circuit, violating the performance requirements of on-the-fly MPC. Moreover, with this pre-processing step the model loses its dynamic nature, where users can compute many different functions on their inputs and can choose these functions dynamically, "on-the-fly". Indeed, using these solutions would limit the parties to only compute functions for which they have already performed the corresponding pre-processing work or computed the corresponding CRS.

[1] For any given family of C, $|C| = \mathsf{poly}(\kappa)$, and thus, $\mathsf{poly}(\kappa, \log(|C|)) = \mathsf{poly}(\kappa)$; but the degree of this polynomial depends on the circuit family.

[2] In our protocol, each party can run Gen in Step 1 and send the vrs to the server in that step. Or in the case of CS proofs, where only a description of a hash function is required, this can be added to the CRS of the protocol.

[3] The protocol has 2 rounds, but (as in the case of SNARGs and SNARKs) the first round is a challenge that is independent of the language and the statement, and can therefore be precomputed by the clients in Step 1 of our protocol. Each challenge can only be used for one proof, so the client must refresh the challenge after each computation.

Verification for Large Inputs

We can make the communication and verification complexities depend merely polylogarithmically on the size of the relevant inputs $x_1, \ldots, x_N$. This requires a succint argument system that is a *proof of knowledge*. This is satisfied by Micali's construction of CS proofs under Valiant's analysis [Mic94, Val08], and by SNARKs [BCCT12, BCCT13]. The complexity of these arguments depends polynomially on the size of the statement being proven, but merely polylogarithmically on the size of the witness for the statement. We thus move c_i from the instance into the witness. To recognize the correct c_i, each party P_i remembers the digest of c_i under a collision-resistant hash function family $\mathcal{H} = \{H_{\mathsf{hk}} : \{0,1\}^* \to \{0,1\}^\kappa\}$.

In the offline stage, every party P_i samples a hash key hk_i and computes the digest $d_i = H_{\mathsf{hk}_i}(c_i)$. Party P_i then sends $(c_i, \pi_i^{\text{ENC}}, \mathsf{hk}_i, d_i)$ to the cloud. Each party P_i remembers its own (hk_i, d_i) pair but can forget the potentially long $x_i, c_i, \pi_i^{\text{ENC}}$. In the online stage, the server broadcasts $(\mathsf{hk}_1, d_1), \ldots, (\mathsf{hk}_N, d_N)$ and proves the following NP statement: "there exist $\widetilde{c}_1, \widetilde{\pi}_1^{\text{ENC}}, \ldots, \widetilde{c}_N, \widetilde{\pi}_N^{\text{ENC}}$ such that $d_i = H_{\mathsf{hk}_i}(\widetilde{c}_i)$ and $c = \mathsf{Eval}(C, (\widetilde{c}_1, \mathsf{pk}_1, \mathsf{ek}_1), \ldots, (\widetilde{c}_N, \mathsf{pk}_N, \mathsf{ek}_N))$ and $\widetilde{\pi}_i^{\text{ENC}}$ is a valid proof".

The construction is secure, since whenever the server convinces the clients, it actually "knows" such $\widetilde{c}_1, \widetilde{\pi}_1^{\text{ENC}}, \ldots, \widetilde{c}_N, \widetilde{\pi}_N^{\text{ENC}}$ which can be efficiently extracted from the server (by the arguments' proof of knowledge property). For an honest party, the extracted $\widetilde{c}_i$ must be the one originally sent by the party (by the collision-resistance of H). For a corrupt party, the extracted $\widetilde{c}_i$ must be a valid ciphertext (by the soundness of $\widetilde{\pi}_i^{\text{ENC}}$) and its plaintext can be efficiently extracted using the secret key used by P_i in the decryption protocol in Step 3.

6.2.4 Formal Protocol

We now write a formal description of our construction of on-the-fly MPC, secure against malicious adversaries, and providing correct verification for large inputs. Our construction requires the following building blocks:

- A semantically-secure multikey fully-homomorphic family of encryption schemes $\mathcal{E} = \left\{\mathcal{E}^{(N)} = (\mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})\right\}_{N>0}$.
- A family of collision-resistant hash functions $\mathcal{H} = \{H_{\mathsf{hk}} : \{0,1\}^* \to \{0,1\}^\kappa\}_{\mathsf{hk}}$.
- A NIZK argument system $\Pi^{\text{ENC}} = (\mathsf{Setup}^{\text{ENC}}, \mathsf{Prove}^{\text{ENC}}, \mathsf{Verify}^{\text{ENC}}, \mathsf{Sim}^{\text{ENC}})$ for the NP relation $R^{\text{ENC}} = \{ \ (\ (\mathsf{pk}, c) \ , \ (x, s) \ ) \ \mid \ c = \mathsf{Enc}(\mathsf{pk}, x \ ; \ s) \ \}$.
- An adaptively extractable SNARK system $\Phi = \left(\mathsf{Setup}^\Phi, \mathsf{Prove}^\Phi, \mathsf{Verify}^\Phi, \mathsf{Ext}^\Phi\right)$ for all of NP.
- An N-party MPC protocol, secure against malicious adversaries corrupting $t < N$ parties, for computing the family of decryption functions

$$g_{c,\mathsf{pk}_1,\mathsf{ek}_1,\ldots,\mathsf{pk}_N,\mathsf{ek}_N}((\mathsf{sk}_1, r_1) \ldots, (\mathsf{sk}_N, r_N))$$
$$\stackrel{\text{def}}{=} \begin{cases} \mathsf{Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_N, c) & \text{if } (\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i) = \mathsf{Keygen}(1^\kappa; r_i) \ \ \forall i \in [N] \\ \bot & \text{otherwise} \end{cases}$$

The protocol is defined as follows:

Input: All parties and the server receive as input the common reference string $\mathsf{crs}^{\text{ENC}}$ for the NIZK proof system Π^{ENC}. If CS proofs are used as the SNARK system, the (description) of the random-oracle hash function is also given to all parties and the server.

Step 1: For $i \in [U]$, party P_i samples a key tuple $(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)$, encrypts its input x_i, and computes a NIZK showing that the ciphertext is well-formed:

$$(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i) := \mathsf{Keygen}(1^\kappa \ ; \ r_i) \quad , \quad c_i := \mathsf{Enc}(\mathsf{pk}_i, x_i \ ; \ s_i)$$

$$\pi_i^{\text{ENC}} \leftarrow \mathsf{Prove}^{\text{ENC}}(\ (\mathsf{pk}_i, c_i)\ ,\ (x_i, s_i)\ )$$

It also samples a hash key hk_i and computes the digest of the ciphertext: $d_i = H_{\mathsf{hk}_i}(c_i)$. It additionally creates a verification reference string and private verification key: $(\mathsf{vrs}_i, \mathsf{priv}_i) \leftarrow \mathsf{Setup}^\Phi(1^\kappa)$.

Party P_i sends the tuple $(\mathsf{pk}_i, \mathsf{ek}_i, c_i, \pi_i^{\text{ENC}}, \mathsf{hk}_i, d_i, \mathsf{vrs}_i)$ to the server, who verifies all proofs $\{\pi_i^{\text{ENC}}\}_{i\in[U]}$.

From this point forward, party P_i can forget its (potentially long) input x_i, ciphertext c_i, and proof π_i^{ENC}. It need only remember its secret key and key-generation randomness (sk_i, r_i), the hash key and digest (hk_i, d_i), and its private verification key priv_i.

A function F, represented as a circuit C, is now selected on inputs $\{x_i\}_{i\in V}$ for some $V \subseteq U$. Let $N = |V|$. For ease of notation, we assume w.l.o.g. that $V = [N]$.

Step 2: The server S computes $c := \mathsf{Eval}(C, (c_1, \mathsf{pk}_1, \mathsf{ek}_1), \ldots, (c_N, \mathsf{pk}_N, \mathsf{ek}_N))$ and creates succinct arguments $\{\varphi_i\}_{i\in[N]}$ for the NP language

$$\begin{aligned} L = \{ \quad \{(\mathsf{pk}_i, \mathsf{ek}_i, \mathsf{hk}_i, d_i)\}_{i\in[N]} \quad | \quad \exists \quad & (\widetilde{c}_1, \widetilde{\pi}_1^{\text{ENC}}) \ , \ \ldots \ , \ (\widetilde{c}_N, \widetilde{\pi}_N^{\text{ENC}}) \ \text{ such that} \\ & d_i = H_{\mathsf{hk}_i}(\widetilde{c}_i) \quad \text{and} \\ & \mathsf{Verify}^{\text{ENC}}(\ (\mathsf{pk}_i, \widetilde{c}_i)\ , \widetilde{\pi}_i^{\text{ENC}}) = 1 \quad \text{and} \\ & c = \mathsf{Eval}(C, (\widetilde{c}_1, \mathsf{pk}_1, \mathsf{ek}_1), \ldots, (\widetilde{c}_N, \mathsf{pk}_N, \mathsf{ek}_N)) \ \} \end{aligned}$$

To compute φ_i, the server uses the verification reference string vrs_i. If CS proofs are used as the SNARK system, the server need only compute a single proof φ that can be verified by all.

The server broadcasts $(c, \varphi_1, \ldots, \varphi_N)$ to all parties $P_1, \ldots, P_N$, together with the tuple $\{(\mathsf{pk}_i, \mathsf{ek}_i, \mathsf{hk}_i, d_i)\}_{i\in[N]}$.

Step 3: Party P_i runs $\mathsf{Verify}^\Phi(\{(\mathsf{pk}_i, \mathsf{ek}_i, \mathsf{hk}_i, d_i)\}_{i\in[N]}, \varphi_i)$ to verify the argument φ_i. If verification is successful for all partiers, they run an MPC protocol ${\Pi_{\text{DEC}}}^{\text{MAL}}$ to compute the function

$$\begin{aligned} & g_{c,\mathsf{pk}_1,\mathsf{ek}_1,\ldots,\mathsf{pk}_N,\mathsf{ek}_N}((\mathsf{sk}_1, r_1) \ldots, (\mathsf{sk}_N, r_N)) \\ & \stackrel{\text{def}}{=} \begin{cases} \mathsf{Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_N, c) & \text{if } (\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i) = \mathsf{Keygen}(1^\kappa; r_i) \ \ \forall i \in [N] \\ \bot & \text{otherwise} \end{cases} \end{aligned}$$

6.2.5 Proof of Security

Theorem 6.2.1. *Let $\mathcal{E}, {\Pi_{\text{DEC}}}^{\text{MAL}}, \mathcal{H}, \Pi^{\text{ENC}}, \Phi$ be as described in Section 6.2.4. Then the above construction is an on-the-fly MPC protocol secure against malicious adversaries corrupting $t < N$ parties and possibly the server S.*

Proof. We prove that the protocol is correct and secure, and that it satisfies the performance requirements of an *on-the-fly* protocol.

Correctness: Correctness follows directly from the correctness properties of homomorphic evaluation and the decryption MPC protocol ${\Pi_{\text{DEC}}}^{\text{MAL}}$.

Performance: The zero-knowledge proofs π_i^{ENC} are independent of C and the size of c is independent of $|C|$ by compactness of homomorphic evaluation. Moreover, the proof φ has size polylogarithmic in $|C|$ and its verification depends only polylogarithmically on the size of the ciphertexts c_i (and therefore polylogarithmically on the size of the inputs x_i as well). Thus, the communication complexity of the protocol is polylogarithmic in $|C|$, and the computation time of each party P_i is at most polylogarithmic in $|C|$ and the total size of the inputs, and polynomial in y and its input x_i.

Security: We show security for the case when the server is corrupted; the case when the server is honest is analogous. Let $\mathcal{A}^{\text{MAL}}$ be a real-world semi-malicious adversary corrupting t clients and the server. Recall that for security, we only need to consider adversaries corrupting a subset T of the parties $P_1, \ldots, P_N$ involved in the computation. Thus, we assume $t < N$, let $T \subsetneq [N]$ be the set of corrupted clients, and let $\overline{T} = [N] \backslash T$.

We construct a simulator $\mathcal{S}^{\text{MAL}}$ as follows. The simulator receives the inputs of the corrupted parties, $\{x_i\}_{i \in T}$ and runs $\mathcal{A}^{\text{MAL}}$ on these inputs $\{x_i\}_{i \in T}$. It simulates the messages for all honest parties in the protocol execution with $\mathcal{A}^{\text{MAL}}$. In Step 1, it samples all key tuples correctly, but encrypts 0 instead of the honest input x_i (which it doesn't know), and computes simulated proofs π_i^{ENC}. In Step 2, it fixes an honest party h and extracts the witness $\{\widetilde{c}_i, \widetilde{\pi}_i^{\text{ENC}}\}_{i \in [N]}$ of the argument φ_h. For all corrupted parties $i \in T$, the simulator extracts the corrupted input $\widetilde{x}_i$ from the proof $\widetilde{\pi}_i^{\text{ENC}}$, submits these to the ideal functionality $\mathcal{F}$, and obtains an output $\widetilde{y}$. In Step 3, it runs the simulator $\mathcal{S}^{\text{MAL}}_{\Pi_{\text{DEC}}}$ for the protocol ${\Pi_{\text{DEC}}}^{\text{MAL}}$, returning $\widetilde{y}$ when it calls the ideal decryption functionality. More formally:

Step 1: The simulator creates the CRS for the NIZK Π^{ENC}, together with a trapdoor key and an extraction key:

$$(\mathsf{crs}^{\text{ENC}}, \mathsf{tk}^{\text{ENC}}, \mathsf{extk}^{\text{ENC}}) \leftarrow \mathsf{Setup}^{\text{ENC}}(1^\kappa)$$

For non-computing parties $i \in \{N+1, \ldots, U\}$ and for honest parties $i \in \overline{T}$, the simulator computes $(\mathsf{pk}_i, \cdot, \mathsf{ek}_i) \leftarrow \mathsf{Keygen}(1^\kappa)$ and samples hk_i honestly. The simulator also runs the verification setup honestly: $(\mathsf{vrs}_i, \mathsf{priv}_i) \leftarrow \mathsf{Setup}^{\Phi}(1^\kappa)$.

The simulator computes an encryption of 0 and simulated zero-knowledge proofs:

$$c_i \leftarrow \mathsf{Enc}(\mathsf{pk}_i, 0) \quad , \quad \pi_i^{\text{ENC}} \leftarrow \mathsf{Sim}(\ \mathsf{tk}^{\text{ENC}}\ ,\ (\mathsf{pk}_i, c_i)\ )$$

It computes the digest $d_i = H_{\mathsf{hk}_i}(c_i)$ honestly. For each party P_i, $\mathcal{S}^{\text{MAL}}$ sends $(\mathsf{pk}_i, \mathsf{ek}_i, c_i, \pi_i^{\text{ENC}}, \mathsf{hk}_i, d_i, \mathsf{vrs}_i)$ to $\mathcal{A}^{\text{MAL}}$ on behalf of P_i.

Step 2: The simulator receives $(c, \varphi_1, \ldots, \varphi_N)$ from $\mathcal{A}^{\text{MAL}}$, together with the tuples $\{(\mathsf{pk}_i, \mathsf{ek}_i, \mathsf{hk}_i, d_i)\}_{i\in[N]}$. The simulator verifies φ_i for all honest parties $i \in \overline{T}$ and for a fixed honest party $h \in \overline{T}$, uses the SNARG extractor to extract witness $\{\widetilde{c}_i, \widetilde{\pi}_i^{\text{ENC}}\}_{i\in[N]}$ from φ_h:

$$\{\widetilde{c}_i, \widetilde{\pi}_i^{\text{ENC}}\}_{i\in[N]} \leftarrow \mathsf{Ext}^{\Phi}\left(\ \{(\mathsf{pk}_i, \mathsf{ek}_i, \mathsf{hk}_i, d_i)\}_{i\in[N]} \ , \ \varphi_h \ \right)$$

It outputs $\perp$ if for any $i \in [N]$, verification fails for φ_i or $\widetilde{\pi}_i^{\text{ENC}}$, or if $d_i \neq H_{\mathsf{hk}_i}(\widetilde{c}_i)$. It also outputs $\perp$ if $c \neq \mathsf{Eval}(C, (\widetilde{c}_1, \mathsf{pk}_1, \mathsf{ek}_1), \ldots, (\widetilde{c}_N, \mathsf{pk}_N, \mathsf{ek}_N))$, or if $\widetilde{c}_i \neq c_i$ for some honest $i \in \overline{T}$.

Step 3: The simulator runs the decryption simulator $\mathcal{S}^{\text{MAL}}_{\Pi_{\text{DEC}}}$ for protocol ${\Pi_{\text{DEC}}}^{\text{MAL}}$ (interacting with $\mathcal{A}^{\text{MAL}}$). When $\mathcal{S}^{\text{MAL}}_{\Pi_{\text{DEC}}}$ queries the ideal decryption functionality with secret key and randomness pairs $\left\{\widetilde{\mathsf{sk}}_i, \widetilde{r}_i\right\}_{i\in T}$, the simulator checks that $\mathsf{Keygen}(1^\kappa \ ; \ \widetilde{r}_i) = (\mathsf{pk}_i, \widetilde{\mathsf{sk}}_i, \mathsf{ek}_i)$ for all $i \in T$. If the check fails, it outputs $\perp$. Otherwise, it decrypts $\widetilde{c}_i$ with the secret key $\widetilde{\mathsf{sk}}_i$ to obtain the corrupted input $\widetilde{x}_i$ (if $\mathsf{Dec}\left(\widetilde{\mathsf{sk}}_i, \widetilde{c}_j\right) = \perp$, it returns $\perp$):

$$\widetilde{x}_i := \mathsf{Dec}\left(\widetilde{\mathsf{sk}}_i, \widetilde{c}_j\right)$$

Finally, it submits inputs $\{\widetilde{x}_i\}_{i\in T}$ to the ideal functionality $\mathcal{F}$, and obtains output $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_N)$, where $\widetilde{x}_i = x_i$ for honest parties $i \in \overline{T}$. It returns $\widetilde{y}$ to the simulator $\mathcal{S}^{\text{MAL}}_{\Pi_{\text{DEC}}}$.

Output: The simulator receives the output of the corrupted parties from $\mathcal{S}^{\text{MAL}}_{\Pi_{\text{DEC}}}$, and returns these as its output.

We prove that $\text{IDEAL}_{\mathcal{F},\mathcal{S}^{\text{MAL}}}(\vec{x}) \overset{c}{\approx} \text{REAL}_{\Pi^{\text{MAL}},\mathcal{A}^{\text{MAL}}}(\vec{x})$ via a hybrid argument.

Hybrid 0: This is a real-world execution of the protocol.

Hybrid 1: We change how Step 3 is performed. Instead of executing the protocol ${\Pi_{\text{DEC}}}^{\text{MAL}}$ where honest parties use their individual secret keys, we run the simulator $\mathcal{S}^{\text{MAL}}_{\Pi_{\text{DEC}}}$ (interacting with $\mathcal{A}^{\text{MAL}}$). When $\mathcal{S}^{\text{MAL}}_{\Pi_{\text{DEC}}}$ queries the ideal decryption functionality with secret keys and randomness $\left\{\widetilde{\mathsf{sk}}_i, \widetilde{r}_i\right\}_{i\in T}$, we return

$$\widetilde{y} = g_{c,\mathsf{pk}_1,\mathsf{ek}_1,\ldots,\mathsf{pk}_N,\mathsf{ek}_N}\left(\left(\widetilde{\mathsf{sk}}_1, \widetilde{r}_1\right)\ldots, \left(\widetilde{\mathsf{sk}}_N, \widetilde{r}_N\right)\right)$$

where $\widetilde{\mathsf{sk}}_i = \mathsf{sk}_i$ and $\widetilde{r}_i = r_i$ for honest parties $i \in \overline{T}$. We define the output of the corrupted parties to be the output of $\mathcal{S}^{\text{MAL}}_{\Pi_{\text{DEC}}}$, and the output of the honest parties to be $\widetilde{y}$.

We claim that Hybrid 0 is computationally indistinguishable from Hybrid 1 by the *security of* ${\Pi_{\text{DEC}}}^{\text{MAL}}$. Indeed, the security of the decryption protocol ${\Pi_{\text{DEC}}}^{\text{MAL}}$ guarantees that as long as we correctly emulate the ideal decryption functionality, the joint output of all parties is computationally indistinguishable in a real-world execution of the protocol with adversary $\mathcal{A}^{\text{MAL}}$ (Hybrid 0), and in an ideal-world execution of the protocol with adversary $\mathcal{S}^{\text{MAL}}_{\Pi_{\text{DEC}}}$ (Hybrid 1). We correctly emulate the ideal decryption functionality, by definition.

Hybrid 2: Hybrid 2 is the same as Hybrid 1 except that we use the extractor Ext^{Φ} to extract a witness $\{(\widetilde{c}_i, \widetilde{\pi}_i^{\text{ENC}})\}_{i\in[N]}$ from φ_h:

$$\{\widetilde{c}_i, \widetilde{\pi}_i^{\text{ENC}}\}_{i\in[N]} \leftarrow \mathsf{Ext}^{\Phi}\left(\ \{(\mathsf{pk}_i, \mathsf{ek}_i, \mathsf{hk}_i, d_i)\}_{i\in[N]} \ , \ \varphi_h \ \right)$$

We define the output of the protocol to be $\perp$ if for any $i \in [N]$, verification fails for $\widetilde{\pi}_i^{\text{ENC}}$ or $d_i \neq H_{\mathsf{hk}_i}(\widetilde{c}_i)$. We also output $\perp$ if $c \neq \mathsf{Eval}(C, (\widetilde{c}_1, \mathsf{pk}_1, \mathsf{ek}_1), \ldots, (\widetilde{c}_N, \mathsf{pk}_N, \mathsf{ek}_N))$, where c is the ciphertext returned by $\mathcal{A}^{\text{MAL}}$ in Step 2. By the *adaptive extractability* property of Φ, we know that this event happens with negligible probability. Therefore, Hybrid 1 and Hybrid 2 are statistically close.

Note that we require Φ to satisfy *adaptive* extractability because the adversary is free to choose the statement of the proof *after* it sees vrs_h.

Hybrid 3: In Hybrid 3, we additionally let the output of the protocol be $\perp$ if $\widetilde{c}_i \neq c_i$ for any honest $i \in \overline{T}$.

We claim that Hybrid 2 and 3 are statistically close by the *collision-resistance* of $\mathcal{H}$. Indeed, Hybrids 2 and 3 are identical except in the case when all previous checks pass but there exists $j \in \overline{T}$ such that $\widetilde{c}_j \neq c_j$. Let ε be the probability, conditioned on all other checks passing, that there exists such a $j \in \overline{T}$. Suppose, for the sake of contradiction, that ε is non-negligible. Then we construct an adversary $\mathcal{B}$ that breaks the collision-resistance of $\mathcal{H}$. The reduction $\mathcal{B}$ works as follows:

1. The reduction chooses arbitrary inputs $\{x_i\}$.
2. It creates the NIZK CRS honestly: $(\mathsf{crs}^{\text{ENC}}, \cdot) \leftarrow \mathsf{Setup}^{\text{ENC}}(1^\kappa)$, and runs $\mathcal{A}^{\text{MAL}}$ on inputs $\{x_i\}_{i\in T}$ and $\mathsf{crs}^{\text{ENC}}$ as the CRS.
3. For all non-computing parties and honest parties, it samples key tuples $(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i) \leftarrow \mathsf{Keygen}(1^\kappa)$, and encrypts the input correctly: $c_i \leftarrow \mathsf{Enc}(\mathsf{pk}_i, x_i \ ; \ s_i)$. It creates honest proofs $\pi_i^{\text{ENC}} \leftarrow \mathsf{Prove}^{\text{ENC}}(\ (\mathsf{pk}_i, c_i) \ , \ (x_i, s_i) \ )$. It also runs the verification setup honestly to generate a verification reference string $(\mathsf{vrs}_i, \cdot) \leftarrow \mathsf{Setup}^{\Phi}(1^\kappa)$.
4. When it receives a hash key hk from the collision-resistance challenger, the reduction guesses an honest index $i^* \leftarrow \overline{T}$ uniformly at random and sets $\mathsf{hk}_{i^*} = \mathsf{hk}$. For all other $i \neq i^*$, it samples hk_i honestly. Finally, for all non-computing and honest parties, it computes the digest $d_i = H_{\mathsf{hk}_i}(c_i)$.
5. It sends $\{\mathsf{pk}_i, \mathsf{ek}_i, c_i, \pi_i^{\text{ENC}}, \mathsf{hk}_i, d_i\}_{i\in\overline{T}}$ to $\mathcal{A}^{\text{MAL}}$.
6. When it receives a ciphertext c and proofs $\varphi_1, \ldots, \varphi_N$ from $\mathcal{A}^{\text{MAL}}$, along with the set $\{\mathsf{pk}_i, \mathsf{ek}_i, \mathsf{hk}_i, d_i\}$, it runs the extractor

$$\{\widetilde{c}_i, \widetilde{\pi}_i^{\text{ENC}}\}_{i\in[N]} \leftarrow \mathsf{Ext}^{\Phi}\left(\ \{(\mathsf{pk}_i, \mathsf{ek}_i, \mathsf{hk}_i, d_i)\}_{i\in[N]} \ , \ \varphi_h \ \right)$$

7. Finally, it submits c_{i^*} and $\widetilde{c}_{i^*}$ to the collision-resistance challenger as its collision.

If all previous checks pass, then in both hybrids we have that $H(c_j) = H(\widetilde{c}_j) = d_j$. Therefore the probability that $\mathcal{B}$ submits a valid collision to the collision challenger is $\varepsilon / \left|\overline{T}\right|$. If ε is non-negligible, then $\mathcal{B}$ breaks the collision-resistance property of the hash family $\mathcal{H}$.

Hybrid 4: In Hybrid 4, we additionally let the output of the protocol be $\perp$ if $\mathsf{Dec}(\widetilde{\mathsf{sk}_i}, \widetilde{c}_i) = \perp$ for any corrupt $i \in T$, where $\widetilde{\mathsf{sk}}_i$ is the secret key output by the decryption protocol simulator $\mathcal{S}^{\text{MAL}}_{\Pi_{\text{DEC}}}$ and $\widetilde{c}_i$ is extracted from the succinct argument φ_h, as in Hybrids 2 and 3.

We claim that Hybrid 3 and Hybrid 4 are statistically close by the *soundness* of the NIZK Π^{ENC}. Indeed, Hybrids 3 and 4 are identical except in the case when all previous checks pass but there exists $j \in T$ such that $\mathsf{Dec}(\widetilde{\mathsf{sk}}_j, \widetilde{c}_j) = \perp$. By *correctness of decryption*, this happens if and only if $\nexists\ (\widetilde{x}_j, \widetilde{s}_j)$ such that $\mathsf{Enc}(\mathsf{pk}_j, \widetilde{x}_j\ ;\ \widetilde{s}_j) = \widetilde{c}_j$, or in other words, if $(\mathsf{pk}_j, \widetilde{c}_j) \notin L^{\text{ENC}}$. Let ε be the probability, conditioned on all other checks passing, that there exists an index $j \in T$ such that $(\mathsf{pk}_j, \widetilde{c}_j) \notin L^{\text{ENC}}$. Suppose, for the sake of contradiction, that ε is non-negligible. Then we construct an adversary $\mathcal{B}$ that breaks the soundness of Π^{ENC}. The reduction $\mathcal{B}$ works as follows:

1. The reduction chooses arbitrary inputs $\{x_i\}$.
2. It receives the CRS from the soundness challenger, and runs $\mathcal{A}^{\text{MAL}}$ on inputs $\{x_i\}_{i \in T}$ and the CRS.
3. For all non-computing parties and honest parties, it samples key tuples $(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i) \leftarrow \mathsf{Keygen}(1^\kappa)$, and encrypts the input correctly: $c_i \leftarrow \mathsf{Enc}(\mathsf{pk}_i, x_i\ ;\ s_i)$. It creates honest proofs $\pi^{\text{ENC}}_i \leftarrow \mathsf{Prove}^{\text{ENC}}(\ (\mathsf{pk}_i, c_i)\ ,\ (x_i, s_i)\ )$. It also runs the verification setup honestly to generate a verification reference string $(\mathsf{vrs}_i, \cdot) \leftarrow \mathsf{Setup}^{\Phi}(1^\kappa)$.
4. It samples hk_i honestly and computes the digest $d_i = H_{\mathsf{hk}_i}(c_i)$.
5. It sends $\{\mathsf{pk}_i, \mathsf{ek}_i, c_i, \pi^{\text{ENC}}_i, \mathsf{hk}_i, d_i\}_{i \in \overline{T}}$ to $\mathcal{A}^{\text{MAL}}$.
6. When it receives a ciphertext c and proofs $\varphi_1, \ldots, \varphi_N$ from $\mathcal{A}^{\text{MAL}}$, along with the set $\{\mathsf{pk}_i, \mathsf{ek}_i, \mathsf{hk}_i, d_i\}$, it runs the extractor
$$\{\widetilde{c}_i, \widetilde{\pi}^{\text{ENC}}_i\}_{i \in [N]} \leftarrow \mathsf{Ext}^{\Phi}\left(\ \{(\mathsf{pk}_i, \mathsf{ek}_i, \mathsf{hk}_i, d_i)\}_{i \in [N]}\ ,\ \varphi_h\ \right)$$
7. It runs the simulator $\mathcal{S}^{\text{MAL}}_{\Pi_{\text{DEC}}}$ (interacting with $\mathcal{A}^{\text{MAL}}$). When $\mathcal{S}^{\text{MAL}}_{\Pi_{\text{DEC}}}$ queries the ideal decryption functionality with secret key and randomness pairs $\left\{\widetilde{\mathsf{sk}}_i, \widetilde{r}_i\right\}_{i \in T}$, it checks that $(\mathsf{pk}_i, \widetilde{\mathsf{sk}_i}, \mathsf{ek}_i) = \mathsf{Keygen}(1^\kappa\ ;\ \widetilde{r}_i)$ for all $i \in [N]$. If this check fails, it returns $\perp$. Otherwise, it chooses a corrupt $i^* \leftarrow T$ uniformly at random and submits $\widetilde{\pi}^{\text{ENC}}_{i^*}$ as its proof forgery.

If all previous checks pass, then in both hybrids we have that $\mathsf{Verify}(\ (\mathsf{pk}_i, \widetilde{c}_i)\ ,\ \widetilde{\pi}^{\text{ENC}}_i\ ) = 1$ for all $i \in [N]$ (see Hybrid 2). Therefore, the probability that $\mathcal{B}$ submits a valid forgery to the soundness challenger is $\varepsilon / |T|$. If ε is non-negligible, then $\mathcal{B}$ breaks the soundness property of the NIZK Π^{ENC}.

Hybrid 5: We now change how we compute $\widetilde{y}$, the value returned to the simulator $\mathcal{S}^{\text{MAL}}_{\Pi_{\text{DEC}}}$ when it queries the decryption ideal functionality. Instead of computing $\widetilde{y} = g_{c, \mathsf{pk}_1, \mathsf{ek}_1, \ldots, \mathsf{pk}_N, \mathsf{ek}_N}\left(\left(\widetilde{\mathsf{sk}}_1, \widetilde{r}_1\right) \ldots, \left(\widetilde{\mathsf{sk}}_N, \widetilde{r}_N\right)\right)$, we first check if $(\mathsf{pk}_i, \widetilde{\mathsf{sk}_i}, \mathsf{ek}_i) = \mathsf{Keygen}(1^\kappa\ ;\ \widetilde{r}_i)$ for all $i \in T$. If this check fails, we return $\perp$; otherwise we decrypt each malicious $\widetilde{c}_i$ and evaluate f on the resulting inputs:

$$\widetilde{y} = \begin{cases} f(\widetilde{x}_1, \ldots, \widetilde{x}_N) & \text{If } (\mathsf{pk}_i, \widetilde{\mathsf{sk}_i}, \mathsf{ek}_i) = \mathsf{Keygen}(1^\kappa\ ;\ \widetilde{r}_i)\ \ \forall i \in T \\ \perp & \text{Otherwise} \end{cases}$$

$$\text{where} \quad \widetilde{x}_i := \mathsf{Dec}(\widetilde{\mathsf{sk}_i}, \widetilde{c}_i) \quad \text{for} \quad i \in T \qquad \text{and} \qquad \widetilde{x}_i = x_i \quad \text{for} \quad i \in \overline{T}$$

We claim that Hybrid 5 and Hybrid 4 are statistically close. In the case when $(\mathsf{pk}_i, \widetilde{\mathsf{sk}_i}, \mathsf{ek}_i) \neq \mathsf{Keygen}(1^\kappa \; ; \; \widetilde{r}_i)$ for some $i \in T$, both hybrids output $\perp$. We focus on the case when this check passes for all parties, so that $\widetilde{\mathsf{sk}}_i$ is guaranteed to be a *valid* secret key for its corresponding public and evaluation keys. In both hybrids, we know that $c = \mathsf{Eval}(C, (\widetilde{c}_1, \mathsf{pk}_1, \mathsf{ek}_1), \ldots, (\widetilde{c}_N, \mathsf{pk}_N, \mathsf{ek}_N))$ (see Hybrid 2). By *soundness* of Π^{ENC}, we know that all $\widetilde{c}_i$'s are fresh encryptions, so by *correctness of multikey evaluation* we know that $\mathsf{Dec}(\widetilde{\mathsf{sk}}_1, \ldots, \widetilde{\mathsf{sk}}_N, c) = f(\widetilde{x}_1, \ldots, \widetilde{x}_N)$, where we define $\widetilde{\mathsf{sk}}_i = \mathsf{sk}_i$ for all honest $i \in \overline{T}$ and $\widetilde{x}_i := \mathsf{Dec}(\widetilde{\mathsf{sk}_i}, \widetilde{c}_i)$ for all $i \in [N]$. Furthermore, since $\widetilde{c}_i = c_i$ for all honest $i \in \overline{T}$ (see Hybrid 3), we know that $\widetilde{x}_i = x_i$ for all $i \in \overline{T}$ by *correctness of decryption*.

Hybrid 6: In Hybrid 6, we change how we compute the proofs π_i^{ENC}. Instead of computing real proofs, we use the NIZK simulator to create simulated proofs:

$$\{\pi_i^{\text{ENC}} \leftarrow \mathsf{Sim}(\; \mathsf{tk}^{\text{ENC}} \; , \; (\mathsf{pk}_i, c_i) \;)\}_{i \in \overline{T}}$$

We claim that Hybrid 6 is computationally indistinguishable from Hybrid 5 by the *unbounded zero-knowledge property* of the proof system Π^{ENC}. Suppose, for the sake of contradiction, that there exists an algorithm $\mathcal{D}$ that distinguishes between hybrids 5 and 6. We construct an adversary $\mathcal{B}$ that breaks zero-knowledge of Π^{ENC}. The reduction $\mathcal{B}$ works as follows:

1. The reduction chooses arbitrary inputs $\{x_i\}$.
2. It receives the CRS from the zero-knowledge challenger, and runs $\mathcal{A}^{\text{MAL}}$ on inputs $\{x_i\}_{i \in T}$ and the CRS.
3. For all non-computing parties and honest parties, it samples key tuples $(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i) \leftarrow \mathsf{Keygen}(1^\kappa)$, and encrypts the input correctly: $c_i \leftarrow \mathsf{Enc}(\mathsf{pk}_i, x_i \; ; \; s_i)$. It creates proofs π_i^{ENC} by calling its oracle with statement (pk_i, c_i) and witness (x_i, s_i). It also runs the verification setup honestly to generate a verification reference string $(\mathsf{vrs}_i, \cdot) \leftarrow \mathsf{Setup}^\Phi(1^\kappa)$.
4. It samples hk_i honestly and computes the digest $d_i = H_{\mathsf{hk}_i}(c_i)$.
5. It sends $\{\mathsf{pk}_i, \mathsf{ek}_i, c_i, \pi_i^{\text{ENC}}, \mathsf{hk}_i, d_i\}_{i \in \overline{T}}$ to $\mathcal{A}^{\text{MAL}}$.
6. When it receives a ciphertext c and proofs $\varphi_1, \ldots, \varphi_N$ from $\mathcal{A}^{\text{MAL}}$, along with the set $\{\mathsf{pk}_i, \mathsf{ek}_i, \mathsf{hk}_i, d_i\}$, it runs the extractor
$$\{\widetilde{c}_i, \widetilde{\pi}_i^{\text{ENC}}\}_{i \in [N]} \leftarrow \mathsf{Ext}^\Phi \left(\; \{(\mathsf{pk}_i, \mathsf{ek}_i, \mathsf{hk}_i, d_i)\}_{i \in [N]} \; , \; \varphi_h \; \right)$$
7. It runs the simulator $\mathcal{S}^{\text{MAL}}_{\Pi_{\text{DEC}}}$ (interacting with $\mathcal{A}^{\text{MAL}}$). When $\mathcal{S}^{\text{MAL}}_{\Pi_{\text{DEC}}}$ queries the ideal decryption functionality with secret key and randomness pairs $\left\{\widetilde{\mathsf{sk}}_i, \widetilde{r}_i\right\}_{i \in T}$, it checks that $(\mathsf{pk}_i, \widetilde{\mathsf{sk}_i}, \mathsf{ek}_i) \neq \mathsf{Keygen}(1^\kappa \; ; \; \widetilde{r}_i)$. If this check fails, it returns $\perp$; otherwise it returns $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_N)$ where $\widetilde{x}_i := \mathsf{Dec}(\widetilde{\mathsf{sk}_i}, \widetilde{c}_i)$ for $i \in T$ and $\widetilde{x}_i = x_i$ for $i \in \overline{T}$.
8. At the end of the protocol, it forwards $\mathcal{A}^{\text{MAL}}$'s output to $\mathcal{D}$ as the output of the corrupt parties, and gives $\widetilde{y}$ to $\mathcal{D}$ as the output of the honest parties.

When $\mathcal{B}$'s oracle is the prover oracle $\mathcal{P}(\cdot)$, then $\mathcal{B}$ perfectly emulates Hybrid 5, whereas if the oracle is the simulation oracle $\mathcal{SIM}_{\mathsf{tk}}(\cdot)$, $\mathcal{B}$ perfectly emulates Hybrid 6. Therefore, if $\mathcal{D}$ can distinguish between Hybrids 5 and 6, then $\mathcal{B}$ breaks the zero-knowledge property of Π^{ENC}.

Hybrids $7.k$ **for** $k = 1, \ldots, N-t$**:** Let $\overline{T} = \{i_1, \ldots, i_{N-t}\}$. In Hybrid $7.k$ we change c_{i_k} so that instead of encrypting x_{i_k} it now encrypts 0. More formally, in Hybrid $7.k$ we have:

$$\left\{c_{i_j} \leftarrow \mathsf{Enc}(\mathsf{pk}_{i_j}, 0)\right\}_{j \le k} \quad , \quad \left\{c_{i_j} \leftarrow \mathsf{Enc}(\mathsf{pk}_{i_j}, x_{i_j})\right\}_{j > k}$$

For ease of notation we let Hybrid 6 be Hybrid 7.0. We claim that the view of $\mathcal{A}^{\text{MAL}}$ in Hybrid $7.k$ is indistinguishable from its view in Hybrid $7.(k-1)$ by the *semantic security of* $\mathcal{E}$ under public key pk_{i_k}. Indeed, now that we run the simulator $\mathcal{S}^{\text{MAL}}_{\Pi_{\text{DEC}}}$ in Step 3 instead of the real decryption protocol, the secret key sk_{i_k} is only used to encrypt c_{i_k}. So suppose, for the sake of contradiction, that there exists an algorithm $\mathcal{D}$ that distinguishes between hybrids $7.k$ and $7.(k-1)$. We construct an adversary $\mathcal{B}$ that breaks the semantic security of $\mathcal{E}$ under public key pk_{i_k}. The reduction $\mathcal{B}$ works as follows:

1. The reduction chooses arbitrary $\{x_i\}$.
2. It creates the NIZK CRS honestly: $(\mathsf{crs}^{\text{ENC}}, \mathsf{tk}^{\text{ENC}}) \leftarrow \mathsf{Setup}^{\text{ENC}}(1^\kappa)$, and runs $\mathcal{A}^{\text{MAL}}$ on inputs $\{x_i\}_{i \in T}$ and $\mathsf{crs}^{\text{ENC}}$ as the CRS.
3. It receives $(\mathsf{pk}, \mathsf{ek})$ from the semantic security challenger and sets $\mathsf{pk}_{i_k} = \mathsf{pk}$ and $\mathsf{ek}_{i_k} = \mathsf{ek}$. Gives $m_0 = 0$ and $m_1 = x_{i_k}$ to the challenger and receives $c = \mathsf{Enc}(\mathsf{pk}, m_b)$. Sets $c_{i_k} = c$. For all $i \in \overline{T}, i \neq i_k$, computes $(\mathsf{pk}_i, \cdot, \mathsf{ek}_i) \leftarrow \mathsf{Keygen}(1^\kappa)$ honestly. For $j < k$, computes $c_{i_j} \leftarrow \mathsf{Enc}(\mathsf{pk}_{i_j}, 0)$ and for $j > k$, computes $c_{i_j} \leftarrow \mathsf{Enc}(\mathsf{pk}_{i_j}, x_{i_j})$.
4. For all non-computing and honest parties, it creates simulated proofs $\pi_i^{\text{ENC}} \leftarrow \mathsf{Sim}(\ \mathsf{tk}^{\text{ENC}}\ ,\ (\mathsf{pk}_i, c_i)\ )$ using the trapdoor tk^{ENC}. It also runs the verification setup honestly to generate a verification reference string $(\mathsf{vrs}_i, \cdot) \leftarrow \mathsf{Setup}^{\Phi}(1^\kappa)$.
5. It samples hk_i honestly and computes the digest $d_i = H_{\mathsf{hk}_i}(c_i)$.
6. It sends $\{\mathsf{pk}_i, \mathsf{ek}_i, c_i, \pi_i^{\text{ENC}}, \mathsf{hk}_i, d_i\}_{i \in \overline{T}}$ to $\mathcal{A}^{\text{MAL}}$.
7. When it receives a ciphertext c and proofs $\varphi_1, \ldots, \varphi_N$ from $\mathcal{A}^{\text{MAL}}$, along with the set $\{\mathsf{pk}_i, \mathsf{ek}_i, \mathsf{hk}_i, d_i\}$, it runs the extractor
$$\{\widetilde{c}_i, \widetilde{\pi}_i^{\text{ENC}}\}_{i \in [N]} \leftarrow \mathsf{Ext}^{\Phi}\left(\ \{(\mathsf{pk}_i, \mathsf{ek}_i, \mathsf{hk}_i, d_i)\}_{i \in [N]}\ ,\ \varphi_h\ \right)$$
8. It runs the simulator $\mathcal{S}^{\text{MAL}}_{\Pi_{\text{DEC}}}$ (interacting with $\mathcal{A}^{\text{MAL}}$). When $\mathcal{S}^{\text{MAL}}_{\Pi_{\text{DEC}}}$ queries the ideal decryption functionality with secret key and randomness pairs $\left\{\widetilde{\mathsf{sk}}_i, \widetilde{r}_i\right\}_{i \in T}$, it checks that $(\mathsf{pk}_i, \widetilde{\mathsf{sk}_i}, \mathsf{ek}_i) \neq \mathsf{Keygen}(1^\kappa\ ;\ \widetilde{r}_i)$. If this check fails, it returns $\perp$; otherwise it returns $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_N)$ where $\widetilde{x}_i := \mathsf{Dec}(\widetilde{\mathsf{sk}_i}, \widetilde{c}_i)$ for $i \in T$ and $\widetilde{x}_i = x_i$ for $i \in \overline{T}$.
9. At the end of the protocol, it forwards $\mathcal{A}^{\text{MAL}}$'s output to $\mathcal{D}$ as the output of the corrupt parties, and gives $\widetilde{y}$ to $\mathcal{D}$ as the output of the honest parties.

When $b = 0$, $\mathcal{B}$ perfectly emulates Hybrid $7.k$, whereas if $b = 1$, $\mathcal{B}$ perfectly emulates Hybrid $7.(k-1)$. Therefore, if $\mathcal{D}$ can distinguish between Hybrids $7.k$ and $7.(k-1)$, then $\mathcal{B}$ can distinguish between an encryption of m_0 and an encryption of m_1, contradicting the semantic security of $\mathcal{E}$.

We have proved that the joint output in Hybrid 0 is computationally indistinguishable from the joint output in Hybrid 7.$(N-t)$. Notice that the joint output in Hybrid 7.$(N-t)$ is precisely $\text{IDEAL}_{\mathcal{F},\mathcal{S}^{\text{MAL}}}(\vec{x})$, and the joint output in Hybrid 0 is defined to be $\text{REAL}_{\Pi^{\text{SM}},\mathcal{A}^{\text{MAL}}}(\vec{x})$. We conclude that $\text{IDEAL}_{\mathcal{F},\mathcal{S}^{\text{MAL}}}(\vec{x}) \overset{c}{\approx} \text{REAL}_{\Pi^{\text{MAL}},\mathcal{A}^{\text{MAL}}}(\vec{x})$, as desired. □

6.2.6 Efficient NIZKs to Prove Plaintext Knowledge

The protocol described in Section 6.2.4 requires a NIZK argument system for the relation NP relation $R^{\text{ENC}} = \{ ((\mathsf{pk}, c) , (x, s)) \mid c = \mathsf{Enc}(\mathsf{pk}, x ; s) \}$. While it is known how to construct NIZK argument systems for all of NP [GOS06, GOS12], using this construction requires expensive NP reductions. In this section, we show how to construct an efficient gap Σ-protocol for R^{ENC} when the encryption scheme is the NTRU-based multikey FHE scheme from Section 5.4. By Theorem 2.2.2 this suffices to construct an efficient NIZK argument system for R^{ENC} in the random oracle model. Our construction follows the ideas of Asharov et al. [AJW11, AJL+12].

Recall that in the aforementioned FHE scheme, a ciphertext has the form $c = [hs + 2e + m]_q$ for public key h, message $m \in \{0, 1\}$, and ring elements s, e, sampled from B-bounded distribution χ. We construct a gap Σ-protocol for proving that "c encrypts 0 under h". That is, we show a protocol for relation

$$R_0^{\text{ENC}} = \left\{ ((h, c) , (s, e)) \mid c = [hs + 2e]_q \ \wedge \ \|s\|_\infty , \|e\|_\infty \leq B \right\}$$

with corresponding language L_0^{ENC}. By Theorem 2.2.1, we can then construct a gap Σ-protocol for R^{ENC} using an OR protocol to prove that "$c \in L_0^{\text{ENC}}$ or $c - 1 \in L_0^{\text{ENC}}$ ".

Gap Σ-protocol for Encryptions of 0. Our construction of a gap Σ-protocol for R_0^{ENC} uses the same parameters as the encryption scheme: degree n, polynomial $\phi(x) = x^n + 1$, modulus q, and distribution $\chi = \overline{D}_{\mathbb{Z}^n, r}$ over the ring $R = \mathbb{Z}[x]/\langle \phi(x) \rangle$. It is additionally parametrized by a distribution $\widetilde{\chi} = \overline{D}_{\mathbb{Z}^n, \widetilde{r}}$ over R, such that $2^{\omega(\log \kappa)} r \leq \widetilde{r} \leq q/4\sqrt{n} - r$. To simplify notation, we recall from Lemma 2.6.4 that χ is B-bounded and $\widetilde{\chi}$ is $\widetilde{B}$-bounded for $B = r\sqrt{n}$ and $\widetilde{B} = \widetilde{r}\sqrt{n}$. By our choice of $\widetilde{r}$, this means that $\widetilde{B} + B \leq q/4$.

To formally describe our protocol, we must first define relations R_{zk} and R_{sound}. We set $B_{\mathsf{zk}} = R_0^{\text{ENC}}$ and set B_{sound} to be essentially the same as R_0^{ENC}, differing only in the requirement set for $\|s\|_\infty$ and $\|e\|_\infty$:

$$R_{\mathsf{sound}} = \left\{ ((h, c) , (s, e)) \mid c = [hs + 2e]_q \ \wedge \ \|s\|_\infty , \|e\|_\infty \leq 4\left(\widetilde{B} + B\right) \right\}$$

Note that since $\widetilde{B} \geq B$, we have $R_{\mathsf{zk}} \subseteq R_{\mathsf{sound}}$. We can now describe our construction:

- $P_1((h, c), (s, e))$: Samples $\widetilde{s}, \widetilde{e} \leftarrow \widetilde{\chi}$ and outputs $a = [h\widetilde{s} + 2\widetilde{e}]_q$ and $st = (\widetilde{s}, s)$.
- $V_1((h, c))$: Outputs a random bit $b \leftarrow \{0, 1\}$.
- $P_2(st, b)$: Parses $st = (\widetilde{s}, s)$ and outputs $z = [\widetilde{s} + bs]_q$.
- $V_2((h, c), a, b, z)$: Computes $\varepsilon = [(a + bc) - hz]_q$ and outputs 1 if and only if $\|z\|_\infty \leq \widetilde{B} + B$, $\|\varepsilon\|_\infty \leq 2\left(B + \widetilde{B}\right)$, and ε is even.

Theorem 6.2.2. *Let $R_{\mathsf{zk}}, R_{\mathsf{sound}}$ be the NP relations described above. The construction $\langle P, V\rangle$ with $P = (P_1, P_2)$ and $V = (V_1, V_2)$ is a gap Σ-protocol for $(R_{\mathsf{zk}}, R_{\mathsf{sound}})$.*

Proof. We show that the above construction satisfies the completeness, special soundness, and HVZK properties.

Completeness: Let $((h, c), (s, e)) \in L_{\mathsf{zk}}$, and let (a, b, z) be a transcript for protocol $\langle P, V\rangle$. Then

$$\varepsilon = [(a + bc) - hz]_q = [h\widetilde{s} + 2\widetilde{e} + bhs + 2be - h\widetilde{s} - hbs]_q = [2(\widetilde{e} + be)]_q = 2(\widetilde{e} + be)$$

where the last inequality holds by the fact that $\widetilde{B} + B \leq q/4$. It is clear that ε is even, and its coefficients are bounded by $2(\widetilde{B} + B)$. Furthermore, $z = \widetilde{s} + bs$, so $\|z\|_\infty \leq \widetilde{B} + B$, as required.

Special Soundness: Let (h, c) be a public key and ciphertext pair, and let $(a, 0, z_0)$ and $(a, 1, z_1)$ be two accepting transcripts. The extractor Ext outputs (s^*, e^*), where $s^* = z_1 - z_0$ and $e^* = [c - hs^*]_q$.

We now argue that $((h, c), (s^*, e^*)) \in R_{\mathsf{sound}}$. By construction, we have that $c = [hs^* + 2e^*]_q$. It remains to show the bound on the size of the coefficients of s^* and e^*. Since $(a, 0, z_0)$ and $(a, 1, z_1)$ are accepting transcripts, we know that $\|z_0\|_\infty, \|z_1\|_\infty \leq \widetilde{B} + B$, so that $\|s^*\|_\infty \leq 2\left(\widetilde{B} + B\right)$.

We now bound e^*. Let $\varepsilon_0 = [a - hz_0]_q$ and $\varepsilon_1 = [(a + c) - hz_1]_q$. Since $(a, 0, z_0)$ and $(a, 1, z_1)$ are accepting transcripts, we know that $\|\varepsilon_0\|_\infty, \|\varepsilon_1\|_\infty \leq 2\left(\widetilde{B} + B\right)$ and both ε_0 and ε_1 are even. Furthermore, $\varepsilon_1 - \varepsilon_0 = [(a + c) - hz_1 - (a - hz_0)]_q = [c - h(z_1 - z_0)]_q = e^*$. This means that e^* is even since both ε_0 and ε_1 are even, and we also have that $\|e^*\|_\infty \leq \|\varepsilon_0\|_\infty + \|\varepsilon_1\|_\infty \leq 4\left(\widetilde{B} + B\right)$, as desired.

Honest-Verifier Zero-Knowledge: Let $((h, c), (s, e)) \in L_{\mathsf{zk}}$ and let $b \in \{0, 1\}$. The simulator Sim chooses $z', e' \leftarrow \widetilde{\chi}$, sets $a' = hz' + 2e' + bc$, and outputs (a', b, z'). We argue that the output of Sim is statistically close to the transcript (a, b, z) of an execution of the protocol $\langle P, V\rangle$. In a real transcript, we have $a = h\widetilde{s} + 2\widetilde{e}$ and $z = \widetilde{s} + \sigma s$. In the simulated transcript, we have $a' = h(z' + bs) + 2(e' + be)$. If $b = 0$, then the distributions are *identical* because $\widetilde{s}, \widetilde{e}, z', e'$ are all sampled from the same distribution $\widetilde{\chi}$. On the other hand, if $b = 1$, then the distributions are *statistically close* by Corollary 2.6.6.

□

Consequences of Having a Gap. We have shown how to construct efficient NIZK arguments for the relation R^{ENC} for the NTRU-based multikey FHE scheme from Section 5.4. However, there is a *gap* in the relations for which soundness and zero-knowledge hold: zero-knowledge holds for an honest prover with a statement in R_{zk}, but an honest verifier is only convinced that the statement is in $R_{\mathsf{sound}} \supseteq R_{\mathsf{zk}}$. We must show that this gap does not affect the correctness of our protocol. It suffices to prove that the scheme is fully homomorphic when the error in fresh ciphertexts is bounded by $B^* \stackrel{\mathsf{def}}{=} 4(\widetilde{B} + B)$.

Our analysis in Section 5.4 does not immediately guarantee this, as it sets $B = \mathsf{poly}(n)$. Since we must have $n = \mathsf{poly}(\kappa)$ for efficiency of the scheme, this means $B = \mathsf{poly}(\kappa)$. However B^* is super-polynomial in κ. Nevertheless, we can easily modify our parameters and analysis to guarantee that the scheme remains fully homomorphic with ciphertext noise that is super-polynomial in κ.

The proof of Lemma 5.4.3 shows that the leveled homomorphic scheme $\mathcal{E}_{\mathsf{LH}}$ described in Section 5.4.2 is multikey homomorphic for N keys and circuits of depth D as long as

$$(nB^*)^{2N+2} < \frac{2^{n^\varepsilon}}{2(8n(nB^*)^{2N+2})^D}$$

which yields the requirement $ND = O\left(n^\varepsilon/(\log n + \log B^*)\right)$. We can then follow the proof of Theorem 5.4.6 and show that there exists a multikey fully homomorphic encryption scheme for $N = O\left(\sqrt{(n^\varepsilon/\log n(\log n + \log B^*))}\right)$. If we set $\widetilde{B} = 2^{\log^2 \kappa} \cdot B$ for $B = \mathsf{poly}(n)$ and $n \geq \kappa$, this is guaranteed if $N = O\left(\sqrt{(n^\varepsilon/\log^3 n)}\right)$ since

$$n^\varepsilon/(\log^3 n) = O(n^\varepsilon/(\log n \cdot (\log n + \log^2 \kappa))) = O(n^\varepsilon/(\log n \cdot (\log n + \log B^*)))$$

(In)Security in the Standard Model. We have shown a NIZK argument for relation R^{ENC}. Though secure in the random oracle model, we remark that care must be taken if we want to hope for security in the standard model. More specifically, since our gap Σ-protocol has only constant soundness, we need to use parallel repetition for soundness amplification. For efficiency, we would like to repeat the protocol only $\mathsf{polylog}(\kappa)$ many times as this already achieves negligible soundness. However, our results in an independent work [DJKL12, BDG+13] (see remarks after Theorem 2.2.2) show that if we use such a small number of repetitions, the resulting NIZK cannot be proven sound (in the standard model) via a black-box reduction to a (super-polynomially hard) falsifiable assumption.

6.3 Impossibility of a 2-Round Protocol

We have shown that there exists an on-the-fly MPC protocol with a 5-round online phase. We now ask whether we can achieve the optimal solution of having a completely non-interactive online phase. In this section we answer this question negatively: we show that the existence of such a protocol (secure against semi-honest adversaries)[4] implies general circuit obfuscation as a virtual black-box with single-bit output, which we know to be impossible [BGI+01]. Our techniques are inspired by those of van Dijk and Jules [vDJ10].

We begin by reviewing the definition of general circuit obfuscation [BGI+01].

Definition 6.3.1 (Circuit Obfuscation [BGI+01])**.** *A probabilistic algorithm $\mathcal{O}$ is a* circuit obfuscator *if the following three conditions hold:*

Functionality: *For every circuit C, the string $\mathcal{O}(C)$ describes a circuit that computes the same function as C.*

Polynomial Slowdown: *There is a polynomial p such that for every circuit C, $|\mathcal{O}(C)| \leq p(|C|)$.*

[4] Considering semi-honest adversaries instead of semi-malicious or malicious adversaries only makes our result stronger.

"Virtual Black-Box" Property: *For any PPT adversary $\mathcal{A}$, there is a PPT simulator $\mathcal{S}$ such that for all circuits C*

$$\left|\Pr[\mathcal{A}(\mathcal{O}(C)) = 1] - \Pr[\mathcal{S}^C(1^{|C|}) = 1]\right| \leq \mathsf{negl}(|C|)$$

Barak et al. [BGI+01] show that assuming one-way functions exist, there does not exist any algorithm $\mathcal{O}$ satisfying Definition 6.3.1, even if we do not require that $\mathcal{O}$ run in polynomial time. Thus, our results imply that assuming one-way functions exist, there does not exist any on-the-fly MPC protocol with a non-interactive online phase.

We now show the connection between on-the-fly MPC and obfuscation. We consider an on-the-fly MPC protocol with a non-interactive online phase, and assume that only one function is evaluated and the function is chosen a-priori, before the start of the protocol (i.e. it does not depend on the offline stage messages). Let N be the number of inputs of the circuit; without loss of generality, we assume that the computing parties are $P_1, \ldots, P_N$. Note that considering such a restricted protocol only makes our impossibility result stronger. A protocol like this can be modeled by efficient and possibly randomized algorithms: $\mathsf{In}_1, \ldots, \mathsf{In}_U, \mathsf{Compute}, \mathsf{Out}_1, \ldots, \mathsf{Out}_N$, where:

- $(d_i, c_i) \leftarrow \mathsf{In}_i(x_i)$: On input x_i, the algorithm In_i outputs two elements, c_i to be sent to the server S and d_i to be kept by party P_i.
- $(z_1, \ldots, z_N) \leftarrow \mathsf{Compute}(C, c_1, \ldots, c_N)$: On input a circuit C and $c_1, \ldots, c_N$, which are the messages the server received from parties $P_1, \ldots, P_N$, $\mathsf{Compute}$ outputs N elements $z_1, \ldots, z_N$. The server sends back z_i to party P_i.
- $y \leftarrow \mathsf{Out}_i(z_i, d_i)$: On input z_i which was received from the server, and the auxiliary information d_i output by In_i, Out_i computes the output y.

We know from the work of Halevi, Lindell, and Pinkas [HLP11] that in the non-interactive setting, the server can always evaluate the circuit multiple times, keeping some parties inputs but plugging in fake inputs of its choosing for the other parties. Thus we must relax the definition of security so that when the server is corrupted, the simulator is allowed to submit queries of the form $(S, \vec{x})$, where S is a non-empty subset of the honest parties and $\vec{x}$ is any input vector of size $n - |S|$. The trusted functionality evaluates the function on $\vec{x}$ and the honest inputs in S. Furthermore, our result holds even when the real-world adversary is only allowed to output 1 bit.[5]

Theorem 6.3.1. *If there exists an on-the-fly MPC protocol with a non-interactive online phase that computes all efficiently computable functions with 2 inputs, and is secure against semi-honest adversaries (with the relaxed definition of security), then there exists a circuit obfuscator $\mathcal{O}$ satisfying Definition 6.3.1.*

Proof. We start by defining a family of "meta-circuits" $\left\{F^{(m)}\right\}_{m \in \mathbb{N}}$. For a fixed $m \in \mathbb{N}$, $F^{(m)}$ is such that given a circuit C of size m and bit-string x, it evaluates C on x and outputs $C(x)$, i.e. $F^{(m)}(C, x) = C(x)$. van Dijk and Juels [vDJ10] show to construct a family of meta-circuits such that for all $m \in \mathbb{N}$, $|F^{(m)}| = O(m^2)$.

[5] Considering a restricted class of adversaries for the on-the-fly MPC protocol only makes our impossibility result stronger.

We now show how to construct a circuit obfuscator $\mathcal{O}$ using an on-the-fly MPC protocol $\Pi = (\mathsf{In}_1, \ldots, \mathsf{In}_U, \mathsf{Compute}, \mathsf{Out}_1, \mathsf{Out}_2)$ with the properties described in the theorem statement. Given a circuit C of size m, $\mathcal{O}$ computes $(\cdot, c_1) \leftarrow \mathsf{In}_1(C)$, samples random coins ρ, σ, τ, and outputs a circuit G that on input x:

- Computes $(c_2, d_2) := \mathsf{In}_2(x\ ; \rho)$.
- Computes $(\cdot, z_2) := \mathsf{Compute}(F^{(m)}, c_1, c_2\ ; \sigma)$
- Computes and outputs $y := \mathsf{Out}_2(z_2, d_2\ ; \tau)$.

We now show that this obfuscator satisfies the functionality, polynomial slowdown, and virtual black-box properties from Definition 6.3.1.

Functionality: The correctness property of the on-the-fly MPC protocol guarantees that $G(x) = F^{(m)}(C, x) = C(x)$ for all x.

Polynomial Slowdown: Using van Dijk and Juel's construction [vDJ10], we have that $|F^{(m)}| = O(m^2)$. Since all algorithms of the on-the-fly MPC protocol run in polynomial time, we have that there exists a polynomial p such that $|G| = p(|C|)$.

Virtual Black-Box: To prove the virtual black-box property, we observe that given an attacker $\mathcal{A}$ trying to break the obfuscation, we can construct a real-world semi-honest adversary $\mathcal{B}$ attacking the on-the-fly MPC protocol, corrupting the server and party P_2. The honest party receives input C and $\mathcal{B}$ receives a dummy value $\widetilde{x}$ for P_2, which it ignores. Instead it receives c_1 from the honest party, builds G as specified and runs $\mathcal{A}$ on G. When $\mathcal{A}$ outputs a bit b, $\mathcal{B}$ completes Steps 2 and 3 in the protocol as specified, and outputs b. We emphasize that any action taken by $\mathcal{A}$ is valid for a semi-honest adversary, so $\mathcal{B}$ is semi-honest.

Security of Π says that there exists simulator $\mathcal{S}$ such that for all inputs $C, \widetilde{x}$, we have $\text{IDEAL}_{\mathcal{F},\mathcal{S}}(C, \widetilde{x}) \overset{c}{\approx} \text{REAL}_{\Pi,\mathcal{B}}(C, \widetilde{x})$, where in the ideal world, $\mathcal{S}$ is given access to an oracle as described above. In the setting we are considering, the only valid subset that $\mathcal{S}$ can provide in a query to this oracle is $\{1\}$. Thus, $\mathcal{S}$ has oracle access to $F^{(m)}(C, \cdot) = C(\cdot)$. We can build a simulator $\mathcal{S}'$ with oracle access to $C(\cdot)$ that on input $|C|$[6], chooses an arbitrary $\widetilde{x}$ and runs $\mathcal{S}(\widetilde{x})$ (which runs $\mathcal{B}$, which runs $\mathcal{A}$), anwers $\mathcal{S}$'s queries with its own oracle, and outputs $\mathcal{S}$'s output.

Since $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs and $\mathcal{S}'$ outputs whatever $\mathcal{S}$ outputs, the fact that $\text{IDEAL}_{\mathcal{F},\mathcal{S}}(C, \widetilde{x}) \overset{c}{\approx} \text{REAL}_{\Pi,\mathcal{B}}(C, \widetilde{x})$ implies that $\mathcal{S}'(|C|) \overset{c}{\approx} \mathcal{A}(G)$. The theorem statement follows.

□

[6] In most applications it is ok to leak the size of the honest input. Indeed this is implied in most constructions, including our construction from Section 6.1.

Chapter 7
Conclusion

In this dissertation, we introduced the notions of cloud-assisted MPC and on-the-fly MPC, modeling the setting in which mutually distrusting parties wish to compute a joint function of their inputs with the aid of a powerful but untrusted server or "cloud". We also showed how to construct cloud-assisted MPC from threshold FHE, which we showed can be obtained from the Ring-LWE assumption. Moreover, we showed how to construct on-the-fly MPC from multikey FHE, a new notion also introduced in this dissertation. We further showed how to construct multikey FHE for any number of keys from the NTRU encryption scheme, based on the DSPR and Ring-LWE assumptions.

Several interesting and important questions remain open. First, is it possible to construct a multikey FHE based solely on the Ring-LWE assumption? In a follow-up work, Bos et al. [BLLN13] combine Brakerki's techniques [Bra12] with our NTRU FHE construction and construct an NTRU-based FHE whose security is based on the Ring-LWE assumption alone. However, their scheme is only multikey for a constant number of keys, which we have shown to be an inherent property of any FHE. A possible approach to solving this problem could be to reduce the DSPR assumption to the Ring-LWE assumption, or to worst-case problems in ideal lattices. Showing a search-to-decision reduction for the DSPR assumption is a possible first step. Another interesting problem that is left open is constructing a multikey FHE scheme that is not "leveled" with respect to the number of keys. Recall that in our construction all algorithms depend on the number of keys, N; removing this dependence remains an open problem. Finally, we are unaware of any other constructions of multikey FHE for any number of keys. Building other multikey FHE schemes, from possibly different assumptions, remains open.

Bibliography

[Abe10] Masayuki Abe, editor. *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*. Springer, 2010.

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Shai Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 595–618. Springer, 2009.

[AD97] Miklós Ajtai and Cynthia Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *STOC*, pages 284–293, 1997.

[AIK10] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *ICALP (1)*, volume 6198 of *Lecture Notes in Computer Science*, pages 152–163. Springer, 2010.

[AJL+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. In Pointcheval and Johansson [PJ12], pages 483–501.

[AJW11] Gilad Asharov, Abhishek Jain, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. *IACR Cryptology ePrint Archive*, 2011:613, 2011.

[Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *FOCS*, pages 106–115. IEEE Computer Society, 2001.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS*, 2012.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive com-

position and bootstrapping for snarks and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *STOC*, pages 111–120. ACM, 2013.

[BD10] Rikke Bendlin and Ivan Damgård. Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In Daniele Micciancio, editor, *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 201–218. Springer, 2010.

[BDG+13] Nir Bitansky, Dana Dachman-Soled, Sanjam Garg, Abhishek Jain, Yael Tauman Kalai, Adriana López-Alt, and Daniel Wichs. Why "fiat-shamir for proofs" lacks a proof. In *TCC*, pages 182–201, 2013.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Paterson [Pat11], pages 169–188.

[BFKL93] Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In Douglas R. Stinson, editor, *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 278–291. Springer, 1993.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In Janos Simon, editor, *STOC*, pages 103–112. ACM, 1988.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Kilian [Kil01], pages 1–18.

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. In *ITCS*, 2012.

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

[BLLN13] Joppe W. Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In Martijn Stam, editor, *IMA Int. Conf.*, volume 8308 of *Lecture Notes in Computer Science*, pages 45–64. Springer,

2013.

[BLV06] Boaz Barak, Yehuda Lindell, and Salil P. Vadhan. Lower bounds for non-black-box zero knowledge. *J. Comput. Syst. Sci.*, 72(2):321–391, 2006.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM Conference on Computer and Communications Security*, pages 62–73. ACM, 1993.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In Safavi-Naini and Canetti [SNC12], pages 868–886.

[BV11a] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In Ostrovsky [Ost11], pages 97–106.

[BV11b] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In Rogaway [Rog11], pages 505–524.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based fhe as secure as pke. In Moni Naor, editor, *ITCS*, pages 1–12. ACM, 2014.

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

[CCK+13] Jung Hee Cheon, Jean-Sébastien Coron, Jinsu Kim, Moon Sung Lee, Tancrède Lepoint, Mehdi Tibouchi, and Aaram Yun. Batch fully homomorphic encryption over the integers. In Johansson and Nguyen [JN13], pages 315–335.

[CDN01] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT*, pages 280–299, 2001.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer, 1994.

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[CKV10] Kai-Min Chung, Yael Tauman Kalai, and Salil P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In Rabin [Rab10], pages 483–501.

[CL08] Giovanni Di Crescenzo and Helger Lipmaa. Succinct np proofs from an extractability

assumption. In Arnold Beckmann, Costas Dimitracopoulos, and Benedikt Löwe, editors, *CiE*, volume 5028 of *Lecture Notes in Computer Science*, pages 175–185. Springer, 2008.

[CLO+13] Ashish Choudhury, Jake Loftus, Emmanuela Orsini, Arpita Patra, and Nigel P. Smart. Between a rock and a hard place: Interpolating between mpc and fhe. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT (2)*, volume 8270 of *Lecture Notes in Computer Science*, pages 221–240. Springer, 2013.

[CLT14] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Scale-invariant fully homomorphic encryption over the integers. In Hugo Krawczyk, editor, *Public Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pages 311–328. Springer, 2014.

[CMNT11] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In Rogaway [Rog11], pages 487–504.

[CNT12] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. In Pointcheval and Johansson [PJ12], pages 446–464.

[Cra12] Ronald Cramer, editor. *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*, volume 7194 of *Lecture Notes in Computer Science*. Springer, 2012.

[DJKL12] Dana Dachman-Soled, Abhishek Jain, Yael Tauman Kalai, and Adriana López-Alt. On the (in)security of the fiat-shamir paradigm, revisited. *IACR Cryptology ePrint Archive*, 2012:706, 2012.

[DKL+13] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure mpc for dishonest majority - or: Breaking the spdz limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS*, volume 8134 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2013.

[DNRS03] Cynthia Dwork, Moni Naor, Omer Reingold, and Larry J. Stockmeyer. Magic functions. *J. ACM*, 50(6):852–921, 2003.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Safavi-Naini and Canetti [SNC12],

pages 643–662.

[DRV12] Yevgeniy Dodis, Thomas Ristenpart, and Salil P. Vadhan. Randomness condensers for efficiently samplable, seed-dependent sources. In Cramer [Cra12], pages 618–635.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[Gam84] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *CRYPTO*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1984.

[Gen09a] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`.

[Gen09b] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009.

[GF05] Harold N. Gabow and Ronald Fagin, editors. *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*. ACM, 2005.

[GGH97] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Public-key cryptosystems from lattice reduction problems. In Burton S. Kaliski Jr., editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 112–131. Springer, 1997.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Johansson and Nguyen [JN13], pages 1–17.

[GGH+13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, pages 40–49. IEEE Computer Society, 2013.

[GGHR14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure mpc from indistinguishability obfuscation. In Yehuda Lindell, editor, *TCC*, volume 8349 of *Lecture Notes in Computer Science*, pages 74–94. Springer, 2014.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Rabin [Rab10], pages 465–482.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In Johansson and Nguyen [JN13], pages 626–645.

[GH11a] Craig Gentry and Shai Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In Ostrovsky [Ost11], pages 107–109.

[GH11b] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. In Paterson [Pat11], pages 129–148.

[GHL+11] Craig Gentry, Shai Halevi, Vadim Lyubashevsky, Christopher Peikert, Joseph Silverman, and Nigel Smart. Personal communication, 2011.

[GHPS12] Craig Gentry, Shai Halevi, Chris Peikert, and Nigel P. Smart. Ring switching in bgv-style homomorphic encryption. In Ivan Visconti and Roberto De Prisco, editors, *SCN*, volume 7485 of *Lecture Notes in Computer Science*, pages 19–37. Springer, 2012.

[GHS12a] Craig Gentry, Shai Halevi, and Nigel P. Smart. Better bootstrapping in fully homomorphic encryption. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2012.

[GHS12b] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In Pointcheval and Johansson [PJ12], pages 465–482.

[GHS12c] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the aes circuit. In Safavi-Naini and Canetti [SNC12], pages 850–867.

[Gil10] Henri Gilbert, editor. *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*. Springer, 2010.

[GK03] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the fiat-shamir paradigm. In *FOCS*, pages 102–113. IEEE Computer Society, 2003.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In Cynthia Dwork, editor, *STOC*, pages 113–122. ACM, 2008.

[GLR11] Shafi Goldwasser, Huijia Lin, and Aviad Rubinstein. Delegation of computation without rejection problem from designated verifier cs-proofs. *Cryptology ePrint Archive:*

Report 2011/456, 2011.

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

[GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *J. Cryptology*, 7(1):1–32, 1994.

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

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for np. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 339–358. Springer, 2006.

[GOS12] Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *J. ACM*, 59(3):11, 2012.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Abe [Abe10], pages 321–340.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO (1)*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92. Springer, 2013.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *STOC*, pages 99–108. ACM, 2011.

[HLP11] Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In Rogaway [Rog11], pages 132–150.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. Ntru: A ring-based public key cryptosystem. In Joe Buhler, editor, *ANTS*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer, 1998.

[HT98] Satoshi Hada and Toshiaki Tanaka. On the existence of 3-round zero-knowledge protocols. In Hugo Krawczyk, editor, *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 408–423. Springer, 1998.

[JN13] Thomas Johansson and Phong Q. Nguyen, editors. *Advances in Cryptology - EURO-*

CRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings, volume 7881 of *Lecture Notes in Computer Science*. Springer, 2013.

[Kal01] Dan Kalman. A Generalized Logarithm for Exponential-Linear Equations. *The College Mathematics Journal*, 32(1), January 2001.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *STOC*, pages 723–732. ACM, 1992.

[Kil95] Joe Kilian. Improved efficient arguments (preliminary version). In Don Coppersmith, editor, *CRYPTO*, volume 963 of *Lecture Notes in Computer Science*, pages 311–324. Springer, 1995.

[Kil01] Joe Kilian, editor. *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*. Springer, 2001.

[KMR11] Seny Kamara, Payman Mohassel, and Mariana Raykova. Outsourcing multiparty computation. Cryptology ePrint Archive, Report 2011/272, 2011. `http://eprint.iacr.org/`.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In Cramer [Cra12], pages 169–189.

[Lip13] Helger Lipmaa. Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT (1)*, volume 8269 of *Lecture Notes in Computer Science*, pages 41–60. Springer, 2013.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Gilbert [Gil10], pages 1–23.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-lwe cryptography. In Johansson and Nguyen [JN13], pages 35–54.

[LTV11] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. Cloud-assisted multiparty computation from fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2011:663, 2011.

[LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Howard J.

Karloff and Toniann Pitassi, editors, *STOC*, pages 1219–1234. ACM, 2012.

[Mic94] Silvio Micali. Cs proofs (extended abstracts). In *FOCS*, pages 436–453. IEEE, 1994.

[MSS13] Steven Myers, Mona Sergi, and Abhi Shelat. Black-box proof of knowledge of plaintext and multiparty computation with low communication overhead. In *TCC*, pages 397–417, 2013.

[Nao03] Moni Naor. On cryptographic assumptions and challenges. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 96–109. Springer, 2003.

[Ost11] Rafail Ostrovsky, editor. *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*. IEEE, 2011.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.

[Pat11] Kenneth G. Paterson, editor. *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*. Springer, 2011.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society, 2013.

[PJ12] David Pointcheval and Thomas Johansson, editors. *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*. Springer, 2012.

[Rab10] Tal Rabin, editor. *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*. Springer, 2010.

[RAD78] R L Rivest, L Adleman, and M L Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation, Academia Press*, pages 169–179, 1978.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Gabow and Fagin [GF05], pages 84–93.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), 2009.

[Rog11] Phillip Rogaway, editor. *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*. Springer, 2011.

[SCO+01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In Kilian [Kil01], pages 566–598.

[SNC12] Reihaneh Safavi-Naini and Ran Canetti, editors. *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*. Springer, 2012.

[SS11a] Peter Scholl and Nigel P. Smart. Improved key generation for gentry's fully homomorphic encryption scheme. In Liqun Chen, editor, *IMA Int. Conf.*, volume 7089 of *Lecture Notes in Computer Science*, pages 10–22. Springer, 2011.

[SS11b] Damien Stehlé and Ron Steinfeld. Making ntru as secure as worst-case problems over ideal lattices. In Paterson [Pat11], pages 27–47.

[SV10] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, 2010.

[SV14] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic simd operations. *Des. Codes Cryptography*, 71(1):57–81, 2014.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2008.

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In Gilbert [Gil10], pages 24–43.

[vDJ10] Marten van Dijk and Ari Juels. On the impossibility of cryptography alone for privacy-preserving cloud computing. In *Proceedings of the 5th USENIX conference on Hot topics in security*, HotSec'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.