Secure Computation with Minimal Interaction

Anat Paskin-Cherniavsky
Secure Computation with Minimal Interaction

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Anat Paskin-Cherniavsky

Submitted to the Senate of
the Technion — Israel Institute of Technology
Elul 5772 Haifa September 2012
The research thesis was done under the supervision of Assoc. Prof. Yuval Ishai and Prof. Eyal Kushilevitz in the Computer Science Department.

List of publications included in this thesis


Another publication made during the Phd. was excluded from this thesis.

I would like to thank several people for making my Phd years very interesting and pleasant. First and foremost, I am endlessly grateful to my advisors, Yuval and Eyal, for their warm and inspiring guidance. I am grateful to them for always finding time to talk (even at very busy times), conversations which often led to simplification or better understanding of months of research. These countless rounds of interaction (;-)) taught me a great deal about precision, exhausting known techniques, and on the power of good abstractions. Most importantly, not to settle for just “a solution that works”, but to look for one as simple as possible, which is “right” for the problem. I consider myself very fortunate for the opportunity to have been their student and learn from them. I am also grateful to Eli Ben-Sasson, for the many interesting complexity theory courses he has given over the years, and for being my temporary advisor for a semester when Yuval and Eyal were away.

My coauthors Amos Beimel, Sigurd Melgard, Claudio Orlandi and Ariel Gallabizon. I’ve learned a lot from them and enjoyed seeing different and creative approaches to problems. Benny Pinkas and Eli Biham for reading an earlier version of this thesis, and providing numerous helpful comments. My friends from the Technion Noga, Elad, Tomer, Gadi, Daniel, Moran, Mor, Omer and others for lunch and coffee breaks full of conversations about research, life and everything. Especially, I would like to thank my roommates Sivan and previously Gal for tolerating me for so long ;-). Oded Lachish for many tips on writing and teaching.

Finally, I want to thank my wonderful family: parents Mira and Arkadi, parents in law Inna and Roman, (practically) sister Anna, and especially grandparents Dina and David for being proud of me. My beloved husband Beni for his friendship and love; for his genuine interest in my work and many insightful comments, and for his great sense of humor. Last but not least, our awesome little son Ofek simply for being here. This thesis is dedicated to them.

The generous financial support of the Technion is gratefully acknowledged.
Contents

Abstract ................................................................. 1
Abbreviations and Notations ...................................... 2

1 Introduction .......................................................... 3
  1.1 Secure computation ................................................. 3
    1.1.1 Feasibility of general MPC .................................. 4
    1.1.2 Round efficient MPC ......................................... 5
      1.1.2.1 MPC without honest majority ......................... 5
      1.1.2.2 MPC with honest majority ................................ 6
  1.2 Results and organization ...................................... 6
    1.2.1 Chapter 3: MPC with minimal interaction ............... 6
      1.2.1.1 Limitations of MPC with minimal interaction ....... 7
      1.2.1.2 Getting around the impossibility results ........... 7
    1.2.2 Chapter 4: On unconditionally secure constant round MPC 9
    1.2.3 Chapter 5: Computing on encrypted data ................. 10

2 Preliminaries ......................................................... 12
  2.1 General ............................................................ 12
  2.2 Randomized encodings of functions .......................... 12
  2.3 Secure multi party computation ................................ 13
  2.4 Representation models ......................................... 16
  2.5 Algebraic notation ............................................... 18

3 On 2-round general MPC ............................................ 19
  3.1 Introduction ....................................................... 19
    3.1.1 Our results .................................................. 20
    3.1.2 Related work ................................................ 21
  3.2 Definitions and tools ............................................. 22
    3.2.1 The PSM model .............................................. 23
    3.2.2 Proof of Theorem 3.1 ...................................... 23
      3.2.2.1 A high level proof .................................. 24
| 3.2.2.2 | Proofs of Lemma 3.1, Lemma 3.2 | 26 |
| 3.2.2.3 | Proof of Lemma 3.3 | 27 |
| 3.2.3 | Secret sharing schemes | 30 |
| 3.2.4 | Set systems | 34 |
| 3.3 | Protocols in the client-server model | 35 |
| 3.3.1 | A \((1, \Omega(n^{1/3}))\)-secure protocol | 36 |
| 3.3.1.1 | Proof of Lemma 3.5 | 38 |
| 3.3.1.2 | Waiving the certified randomness assumption | 39 |
| 3.3.2 | An \(n = 3t + 1\) server \((1, t)\)-secure construction (inefficient) | 42 |
| 3.3.3 | An \(n = O(t^2/\log(t))\) server \((1, t)\)-secure construction (efficient) | 44 |
| 3.4 | Full security for \(t = 1\) | 49 |
| 3.4.1 | Proof of Theorem 3.10 | 51 |
| 3.5 | Security with selective abort | 53 |
| 3.5.1 | A private protocol with knowledge of outputs (step 4) | 53 |
| 3.5.2 | From privacy with knowledge of outputs to security with selective abort (step 2) | 57 |
| 3.5.3 | Technical details for the (simplified) construction | 58 |
| 3.5.3.1 | A simple private protocol for degree-2 functionalities | 58 |
| 3.5.3.2 | A private protocol for degree-\(d\) polynomials | 58 |
| 3.5.3.3 | Unconditional MACs | 62 |
| 3.5.3.4 | Moving from degree-3 polynomials to general functions (step 3) | 62 |
| 3.5.3.5 | MCDS for the (generalized) client-server setting | 64 |
| 3.5.3.6 | More details for step 3 | 66 |
| 3.5.4 | Making black-box use of arbitrary PRG | 68 |
| 4 | Limitations on constructing perfect constant-round protocols | 74 |
| 4.1 | Introduction | 74 |
| 4.1.1 | Related work | 75 |
| 4.1.2 | Overview of techniques | 76 |
| 4.2 | Preliminaries | 79 |
| 4.2.1 | Polynomials | 79 |
| 4.2.2 | Parallel algorithms | 79 |
| 4.3 | Some required algebra | 80 |
| 4.3.1 | Linearized polynomials | 80 |
| 4.3.2 | Properties of quadratic forms | 80 |
| 4.3.3 | On distributions of alternate matrices | 83 |
| 4.4 | Root counting for odd characteristic | 87 |
| 4.4.1 | Quadratic forms | 87 |
| 4.4.2 | General degree-2 polynomials | 88 |
| 4.5 | Root counting and randomizing polynomials | 90 |
| 4.6 | Applications to parallel algorithms | 91 |
| 4.7 | Root counting for characteristic 2 | 93 |
| 4.7.1 | Notation and background | 94 |
### 4.7.2 A root-counting algorithm for quadratic forms

- **4.7.2.1** Simplification of $Q$ with $n = 2^k$ ................................. 95
- **4.7.2.2** Simplification of general $Q$ ............................................ 97

### 4.7.3 A root counting algorithm for general polynomials .................. 98

---

## 5 Evaluating branching programs on encrypted data ........................ 100

- **5.1** Introduction ............................................................................. 100
  - **5.1.1** Our contribution .............................................................. 102

- **5.2** Preliminaries ........................................................................... 104

- **5.3** Cryptographic primitives ....................................................... 105
  - **5.3.1** Computing on encrypted data ......................................... 105
  - **5.3.2** Oblivious transfer ............................................................ 107

- **5.4** Main protocol ......................................................................... 109
  - **5.4.1** Implementing strong OT .................................................. 112
  - **5.4.2** Optimizations .................................................................. 114

- **5.5** Handling malicious clients .................................................... 115
  - **5.5.1** Trusted setup model .......................................................... 116
  - **5.5.2** Fully malicious model .................................................... 118
    - **5.5.2.1** An intermediate conditional privacy result ................. 123
    - **5.5.2.2** The final construction ............................................. 125

---

Abstract in Hebrew

- **Abstract in Hebrew** ..................................................................... 8
Abstract

In the problem of secure multiparty computation (MPC) we have \( n \) parties who want to jointly evaluate a function \( f \) on their local inputs. An MPC protocol should allow the parties to correctly compute \( f \) while hiding the inputs from each other to the extent possible. An important complexity measure of MPC protocols is their round complexity. In this thesis, we study a few questions related to the goal of minimizing the round complexity of MPC.

2-round MPC. We study the possibility of MPC with only two rounds of interaction in the case that an honest majority is assumed. We obtain several positive results which complement previous negative results in the area.

On constant-round unconditional MPC. We show that a common general technique for constructing such protocols can apply only to functions \( f \) in the class \( \text{NC} \) (that is, functions efficiently computable in parallel).

Many previous constant-round protocols relied on efficient representations of functions by low-degree randomizing polynomials. In most of these representations, the degree in the randomness is at most 2 and the degree in the input is constant. We show that any function \( f \) which has an efficient representation with such degree constraints must be in \( \text{NC} \).

On the flip side, this negative result provides an avenue for obtaining new parallel algorithms via the construction of randomizing polynomials. We provide several examples for the potential usefulness of this approach.

Computing on encrypted data. We present a 2-message two-party protocol for evaluating branching programs such that the communication complexity depends only on the length of the branching program and not on its size. This protocol implies an encryption scheme which supports an efficient evaluation of bounded-length branching programs on encrypted data. Subsequently to our work, a more general result (allowing to evaluate arbitrary circuits on encrypted data) was given in a breakthrough result of Gentry. However, our protocol is simpler, relies on a different intractability assumption, and can be more easily modified to offer protection against malicious parties.
Abbreviations and Notations

MPC — Secure Multi Party Computation
PSM — Private Simultaneous Messages
RP — Randomizing Polynomials
TP — Trusted Party
Chapter 1

Introduction

The main goal of this research is to study the feasibility and efficiency of secure computation with a small amount of interaction. Before discussing the concrete technical questions we intend to study, we give some general background to put them in the proper context.

1.1 Secure computation

The following description of MPC is informal, see Chapter 2, and further [26] for formal definitions and discussion. In secure multi-party computation (MPC), we have \( n \geq 2 \) parties who want to jointly evaluate a (publicly known) function \( f \) of their joint input\(^1\). The computation is to be carried out by a protocol, proceeding in rounds. In each round, parties send (and receive) messages to (from) other parties over a network. At the end of the execution, all parties output a value. On a very intuitive level, a secure protocol should hide the parties’ inputs from other parties, except for information that the output of \( f \) combined with their local input reveals [124, 61, 14, 28]. Additionally, it should guarantee that the outputs of honest parties are consistent with their local inputs.

These security requirements should hold even in the presence of an adversary that fully controls some subset of the parties, and makes them behave arbitrarily. We consider adversaries controlling up to \( t \) parties, for some \( t < n \). Unless explicitly stated otherwise, we consider protocols with polynomially bounded parties, and thus limit ourselves to questions on evaluating functions in POLY (that is, functions that can be computed efficiently disregarding security issues).

In the following, we briefly describe the model of secure computation we use in this work.

Network model. The network consists of point-to-point channels between each pair of parties, such that all other parties can neither eavesdrop nor modify information on the link. The existence of a broadcast channel, allowing each party to send the same message to all other parties is sometimes assumed. Even if

\(^1\)In the context of MPC a “function”, also referred to as a “functionality”, is a (possibly randomized) mapping from \( n \) inputs to \( n \) outputs.
the sender is corrupted, a broadcast channel guarantees that all honest parties receive the same message. We consider the standard synchronous model, where each round starts at the tick of a centralized clock, at which point all parties send their messages, based on their input, randomness, and messages received in previous rounds. However, the adversary has the capability of rushing, in the sense that it can first hear the round messages of the other parties, and only then send its own messages (which can depend on the messages it heard in this round).

**Notion of security.** We consider standard, simulation-based security. Intuitively, we want the adversary to cause “no more damage” than in an ideal-world process, where there is a trusted third party TP, to which all parties send their inputs. TP evaluates $f$, and sends back a reply to all parties (essentially, the adversary can cause “no more damage” than modifying its local input independently of the other inputs). More precisely, we say that a protocol is perfectly (resp., statistically, computationally) $t$-secure for $t < n$, if for any real-world process adversary corrupting at most $t$ parties, there exists an ideal process adversary corrupting the same set of parties, such that the output of the real process is perfectly (resp., statistically, computationally) indistinguishable from the output of the ideal process. Here the “output” of a process consists of the output of the adversary together with the output of the uncorrupted parties.

**Common relaxations of the security notion.** Sometimes, full security in the above sense cannot be achieved, and certain relaxations of the above model, considering certain “security breaches” as valid, are made. One common relaxation is to allow the adversary to abort the protocol. There are several flavors of security with abort. The standard notion from the literature allows the adversary to learn the output, and then decide whether the honest parties learn the output, or learn nothing. Another common relaxation, referred to as the semi-honest model, captures the case of a passive adversary who tries to learn additional information without modifying the behavior of corrupted parties. This is formalized by assuming that in both the real and the ideal process the adversary cannot change the behavior of corrupted parties except for modifying their output. We refer to an adversary not assumed to be semi-honest (which is the default) as malicious. See [60] for more details on the various relaxations above.

### 1.1.1 Feasibility of general MPC

The qualitative feasibility question of general MPC is essentially resolved. Next, we briefly describe the concrete state of affairs in general MPC.

**MPC with honest majority.** In the case of honest majority, any function (in POLY) can be evaluated with perfect security for $t < n/2$ semi-honest parties, and $t < n/3$ malicious parties assuming secure point-to-point channels but not broadcast [14, 28]. If perfect security is required, the latter resilience threshold is tight [14] even assuming broadcast. If broadcast is allowed, and settling for statistical security, there exists statistically secure general MPC against $t < n/2$ malicious parties [15].

**MPC without honest majority.** Statistical security is generally impossible with no honest majority, even in the semi-honest model [33]. The same holds for computational security without abort [35]. Settling for computational security with abort, general secure computation without honest majority is possible assuming
the existence of oblivious transfer (OT) [124, 61, 60]. Observe that the notion of security with abort does not guarantee “fairness” in the sense that it allows the adversary to learn the outputs of corrupted parties while preventing honest parties from learning their outputs.

### 1.1.2 Round efficient MPC

Minimizing the round complexity of secure computation in various settings has been the subject of intense study. In the following, we briefly discuss previous work on minimizing round complexity of MPC protocols in various settings.

#### 1.1.2.1 MPC without honest majority

**The two-party case.** In two-party protocols, the typical convention is that only one party speaks in each round. Our entire discussion of this case is assuming this convention. Additionally, we refer to functionalities where only one party receives output (protocols for such functionalities can be turned into general two-party protocols using a single additional round). In the two-party setting, 2-round protocols (in different security models and under various setup assumptions) were given in [124, 118, 24, 70, 29]. Constant-round two-party protocols with security against malicious parties were given in [98, 88, 100, 81, 79]. In [88] it was shown that the optimal round complexity for secure two-party computation without setup is 5. This is essentially tight, in the sense that the negative result is restricted to protocols with black-box simulation.

**Computing on encrypted data.** In this setting secure two-party computation, involving a sender and a receiver is considered, concentrating on the semi-honest model. Here the sender’s input is an encoding of a function $f$, the receiver’s input is a string $x$, and the receiver learns $f(x)$. Recall that in the two-party setting we by default consider functionalities where only one party (in this case, the “receiver”) gets an output, and protocols where only one party speaks in each round. So, in this setting minimal interaction means one message from the receiver to the sender followed by one message from the sender to the receiver. Such protocols can be thought of as computing on encrypted data by viewing the receiver’s messages as an “encryption” of the input, on which the sender should evaluate a function without further interaction with the receiver. The Sender’s reply is viewed as an “encryption” of the output, to be “decrypted” by the receiver.

Besides minimal interaction, another essential requirement is to minimize receiver’s work (computation, and incoming/outgoing communication), trying to get as close as possible to $|x| + |f(x)|$, which is obtained without security. More concretely, we require that receiver’s work is sublinear in the sender’s input, which can be much longer than $x$ (ideally independent of it). On the sender’s side, protocols of this kind grow more efficient as the representation model of $f$ becomes more succinct. For instance, truth tables is a very wasteful representation, with representation size $\exp(|x|)$ for all functions, while circuits is the most efficient representation one can hope for (making sender’s work polynomial in the work required without security).

Some previously known constructions can be cast as protocols for computing on encrypted data for various representation models. For instance, 2-round PIR [72, 25] protocols imply computing on encrypted data for truth tables, where in [25] receiver’s work is independent of server’s input size. The work of [54] can be cast as computing on encrypted data for sets, slightly improving the representation model. No protocols for much stronger representation models were known. For example, although [124]’s protocol seems to fit the framework’s “structural” specification for circuits, it does not satisfy the main requirement of receiver’s
work being sublinear in the sender’s input. In particular, the protocols’ communication complexity linearly depends on the circuit size of $f$.

As mentioned above, computing on encrypted data for circuits is the holy grail in the line of research on computing on encrypted data. One way to obtain it, is by obtaining a homomorphic cryptosystem. Loosely speaking, a homomorphic cryptosystem allows to both add and multiply ciphertexts. The question of whether such cryptosystems exist remained open for several decades, with no substantial progress. Quite surprisingly, a plausible candidate for a homomorphic encryption was recently put forward by Gentry [57]. More specifically, in [57] they propose a homomorphic cryptosystem based on a non-standard hardness assumption on lattices. This seminal work has been followed by a long line of work in the last few years, focusing both on relying on more standard assumptions, and improving practical efficiency of the scheme [23, 45, 58].

**The multi-party case.** Constant round protocols for the general case with no honest majority have been devised in [89, 112], under various computational assumptions. These protocols do not attempt to optimize the concrete number of rounds required.

### 1.1.2.2 MPC with honest majority

More relevant to our research is previous work on the round complexity of MPC with honest majority and guaranteed output delivery. In this setting, constant-round protocols were given in [6, 11, 10, 9, 73, 55, 36, 74, 38, 41, 85, 86]. In particular, it was shown in [55] that 3 rounds are sufficient for general MPC with $t = \Omega(n)$ malicious parties, where one of the rounds requires broadcast. This result is obtained with computational security for all functions, and with unconditional security for functions in “low” complexity classes, which have efficient branching programs. The question of minimizing the exact round complexity of MPC over point-to-point networks was explicitly considered in [85, 86]. In contrast to (one of) our research directions, the focus of these works is on obtaining nearly optimal resilience.

### 1.2 Results and organization

In this thesis, we have obtained a few results on questions related to MPC with few rounds of interaction. Chapter 2 includes definitions and notation used throughout the thesis. The three subsequent chapters describe each of our three main results. In this section, we present motivation and a summary of each of these results in the order of their appearance in the thesis.

#### 1.2.1 Chapter 3: MPC with minimal interaction

**Motivating example.** Consider the following motivating scenario. Two or more employees wish to take a vote on some sensitive issue and let their manager only learn whether a majority of the employees voted “yes”. Given an external trusted server, we have the following minimalistic protocol: each employee sends her vote to the server, who computes the result and sends it to the manager.

When no single server can be completely trusted, one can employ an MPC protocol involving the employees, the manager, and (possibly) additional servers. A practical disadvantage of MPC protocols from
the literature that offer security against malicious parties is that they involve a substantial amount of interaction. This interaction includes three or more communication rounds, of which at least one round consists of broadcast messages.

1.2.1.1 Limitations of MPC with minimal interaction

We study whether it is possible to obtain protocols with only two rounds of interaction, which resemble the minimal interaction pattern of the centralized trusted server solution described above. That is, we allow to employ several untrusted servers instead of a single trusted server, and require the same minimal interaction pattern. More generally, we consider general MPC in a client-server setting, where there are \( m \geq 2 \) clients some of whom hold inputs and some of whom should receive outputs (not necessarily disjoint), and \( n \) additional servers with no inputs and outputs. The communication pattern is such that each client sends a single message to each server and each server sends a single message to each client who should receive output.

Back to the more standard MPC setting, where all parties may contribute inputs and expect to receive outputs, the corresponding goal is to obtain general MPC protocols which involve only two rounds of point-to-point communication between the parties and guarantee output delivery to all parties. The answer to the above question in the semi-honest model is quite well-understood and optimistic. Namely, there exist 2-round general MPC protocols with resilience \( n > 3t \) ([11] combined with [73]), and 3-round general MPC protocols with resilience \( n > 2t \) ([11] combined with [14]). The above protocols are unconditionally (perfectly or statistically) secure for functions in low complexity classes such as \( \text{NC}^1 \), and computationally secure for \( f \in \text{POLY} \) (making a black-box use of a pseudorandom generator (PRG)). On the other hand, this goal may seem too ambitious for the malicious setting. In particular:

- Broadcast is a special case of general MPC, and implementing broadcast over secure point-to-point channels generally requires more than two rounds [52].
- Even if a free use of broadcast messages is allowed in each round, it is known that three or more communication rounds are necessary for general MPC protocols which tolerate \( t \geq 2 \) corrupted parties, regardless of the total number of parties [56].

1.2.1.2 Getting around the impossibility results

We complement the negative results above on 2-round MPC, with matching positive results. More concretely, we explore two directions left open by these negative results. One direction is understanding whether full security can be achieved for the case \( t = 1 \). Another is understanding which meaningful relaxations of full security can be achieved while tolerating \( t > 1 \) malicious parties. As mentioned above, security against semi-honest adversaries is a known such relaxation. We explore generalizations of this result by studying the possibility of achieving (some flavor of) security with abort, that is, where output delivery is not guaranteed.

\[^2\]Alternatively, broadcast rounds can be emulated using point-to-point communication. However, this further increases the round complexity of the protocol. See [85] for discussion.
The case \( t = 1 \). For starters, neither of the above limitations rules out the possibility of 2-round general MPC in the case of a single corrupted party with full security. That is, the following question is raised:

**Question 1.1.** Are there general MPC protocols that resist a single malicious party and require only two rounds of communication over point-to-point channels?

This question may be highly relevant to typical real-world situations where the number of parties is small and the existence of (even) two corrupted parties is unlikely. The question is open even if broadcast messages are allowed in each round. Recall that general MPC with 3 rounds when broadcast is assumed is known to be possible by [55]. Furthermore, since their protocol uses broadcast only in one round, and for \( t = 1 \) broadcast can be easily implemented in 2 rounds, this yields 4-round protocols for \( t = 1 \).

Our first main result is a strong affirmative answer to Question 1.1. Namely, we prove that there exists a general 2-round MPC protocol for \( n \geq 5 \) parties which is secure against a single malicious party. The protocol can provide statistical security for a class of “computationally simple” functions (which includes \( \text{NC}^1 \)) and computational security for arbitrary functions by making a black-box use of a pseudorandom generator. We also prove a similar result in the client-server setting. We have several variants of the construction, with different efficiency/resilience tradeoffs.

- A secure protocol tolerating either a single corrupted client or coalitions of \( t < n/3 \) corrupted servers (but not both). We refer to this kind of protocols as \((1, t)\)-secure. The complexity of this protocol grows linearly with \( \binom{n}{t} \), so it is not efficient in the number of servers, and is only applicable for small \( n \). All other protocols we have constructed are efficient in all parameters, including \( n \).

- A \((1, t)\)-secure protocol for \( t = \Theta((n \cdot \log(n))^{1/2}) \) (this construction is efficient in \( n \)).

Similarly to the main result in the standard setting, the latter protocols for the client-server setting provide computational security for \( f \in \text{POLY} \) by making a black-box use of a pseudorandom generator, and statistical security for \( f \in \text{NC}^1 \). We note that we can even achieve perfect \((1, t)\)-security for \( f \in \text{NC}^1 \) with a smaller resilience of \( t = \Theta(n^{1/3}) \).

**Relaxed notion of security.** Another possibility left open by the above negative results is to tolerate \( t > 1 \) malicious parties by settling for a weaker notion of security against malicious parties. As mentioned in Section 1.1, a common relaxation is to allow security with abort.

Unfortunately, this notion of security is not liberal enough to get around the first negative result. But it turns out that a further relaxation of this notion, which we refer to as security with selective abort, is not ruled out by either of the above negative results. This notion, introduced in [64], differs from the standard notion of security with abort in that it allows the adversary (after learning its own output) to individually decide for each uncorrupted party whether this party will obtain its correct output or \( \bot \). Indeed, this mild relaxation turned out to be useful, and [64] shows that two rounds of communication over point-to-point channels are sufficient to realize broadcast under this notion, with an arbitrary number of corrupted parties. This gives rise to the following question:

---

3 As mentioned above, the first protocol is only efficient for sufficiently small \( n \).

4 Mapping our terminology to that of [64], our notions of “security with abort” and “security with selective abort” correspond to the notions of “security with unanimous abort and no fairness” and “security with abort and no fairness” from [64]. We note that the negative result from [56] can be extended to rule out the possibility of achieving fairness in our setting with \( t > 1 \).
**Question 1.2.** Are there general MPC protocols that require only two rounds of communication over point-to-point channels and provide security with selective abort against $t > 1$ malicious parties?

Our second main result is an affirmative answer to Question 1.2 (for general MPC), via a protocol with linear resilience. More specifically, we obtain a general 2-round MPC protocol for $n > 3t$ which is $t$-secure with selective abort against $t$ malicious parties. Similarly to our previously described results, the protocol is efficient with statistical security for $f \in \text{NC}^1$, and with computational security for $f \in \text{POLY}$. However, in the computational case the protocol relies on a stronger cryptographic assumption - the existence of a PRG in $\text{NC}^1$, and furthermore makes a non-black-box use of such a PRG.

### 1.2.2 Chapter 4: On unconditionally secure constant round MPC

Another problem we study, still focusing on the setting with honest majority, somewhat deviates from the quest for a minimal communication pattern and asks what functions can be computed in a constant number of rounds. As mentioned in Section 1.1.2.2, this question is positively resolved for all efficiently computable functions (the best one can hope for), in the computational setting. Here constant-round protocols tolerating a linear fraction of maliciously corrupted parties are known. On the other hand, the following remains a big open question.

**Question 1.3.** Do constant-round, unconditionally secure protocols with polynomial communication complexity exist for all of $\text{POLY}$? Specifically, what can be said about perfectly secure protocols?

The above question is open even in the semi-honest setting. Theoretically, such protocols are interesting, as they would allow for efficient MPC not relying on unproven assumptions. The question is also of practical importance, since perfectly secure protocols tend to be more efficient than computationally (and even statistically) secure ones, as their complexity does not depend on a security parameter. Our main result in this direction is a negative one, demonstrating that a common general technique for designing perfectly secure constant round MPC cannot be applied to obtain such protocols for functions outside of $\text{NC}$.

Randomizing polynomials (RP) represent a function $f(x)$ by a randomized mapping $p(x, r)$ over a finite field $\mathbb{F}$ such that, for any input $x$, the output distribution of $p(x, r)$ depends only on the value of $f(x)$. We study the class of functions $f$ which admit an efficient representation by constant-degree RP. It is known that this class contains $\text{NC}^1$ as well as log-space classes contained in $\text{NC}^2$. Whether it contains all polynomial-time computable functions is a wide open question. A positive answer would have major and unexpected consequences, including a positive answer to Question 1.3.\(^5\)

We obtain evidence for the limited power of RP by showing that a useful subclass of constant-degree RP cannot efficiently capture functions beyond $\text{NC}$. Concretely, we consider RP over fields $\mathbb{F}$ of a small characteristic in which each monomial has degree (at most) 2 in the random inputs $r$ and constant degree in $x$. This subclass captures most constructions of randomizing polynomials from the literature. We show that all functions $f$ which can be efficiently represented by such randomizing polynomials over fields of a small characteristic are in non-uniform $\text{NC}$. The same result holds over arbitrary fields given a quadratic residuosity oracle. This result is obtained in two steps: (1) we observe that computing $f$ as above reduces to counting roots of degree-2 multivariate polynomials; (2) we design parallel algorithms for the latter problem. These parallel root counting algorithms may be of independent interest.

\(^5\)An additional consequence is the equivalence of (polynomial-time) cryptography and cryptography in $\text{NC}^0$.  

---

9
On the flip side, our main result provides an avenue for obtaining new parallel algorithms via the construction of RP. This gives an unexpected application of cryptography to algorithm design. We provide several examples for the potential usefulness of this approach.

1.2.3 Chapter 5: Computing on encrypted data

In this study, we consider the model of computing on encrypted data as described in Section 1.1.2. Namely, secure two-party computation, involving a sender and a receiver is considered, concentrating on the semi-honest model. Here the sender’s input is an encoding of a function $f$, the receiver’s input is a string $x$, and the receiver learns $f(x)$. The goal is to devise a 2-message protocol for the task, with communication sublinear in the sender’s input (ideally independent of it). Such protocols can be thought of as computing on encrypted data by viewing the receiver’s messages as an “encryption” of the input, on which the sender should evaluate a function without further interaction with the receiver. The sender’s reply is viewed as a short “encryption” of the output, to be “decrypted” by the receiver.

On the sender’s side, protocols of this kind grow more efficient as the “encoding scheme” (subsequently referred as a “representation model”) of $f$ becomes more “succinct”. For example, circuits are the most succinct encoding one can hope for, in the sense that if $f$ is efficiently computable, then it has polynomial-sized circuits evaluating it (by definition).

We present a protocol for evaluating length-bounded branching programs on encrypted data. More specifically, receiver’s work polynomially depends on its input, and on the length of $f$’s branching program. As interesting special cases, one can efficiently evaluate finite automata, decision trees, and OBDDs (Ordered Binary Decision Diagrams) on encrypted data. Our construction is based on an additively homomorphic cryptosystem with certain properties [43]. Informally, an additively homomorphic (public key) cryptosystem is a cryptosystem allowing to compute the encryption of the “sum” of two plaintexts given only encryptions of the plaintexts and the public key (examples of such cryptosystems are [65, 43] and several more).6 We also show how to strengthen the above result to work against malicious receivers, in the sense that even if the receiver sends an invalid encryption, it cannot learn more information about the sender’s branching program than is possible by providing some valid encrypted input.

Our work may still be of some relevance even in light of homomorphic schemes proposed in [57] and subsequent work, which positively resolves the question for the representation model of circuits. However, our protocol relies on a different (and seemingly more conservative) intractability assumption, and can be more easily modified to offer protection against malicious parties. In some more detail, we want to make the protocol robust against a malicious receiver, in the sense that it only learns $f(x^*)$ for some “effective” $x^*$, independent of $f$, even if it submits a malformed “encryption”. The problem is that it may be unclear how to efficiently verify that the encryption is indeed well-formed. One approach is to let the receiver prove that his encryption is well-formed using a non-interactive zero knowledge proof (NIZK). Such proofs are useful in this setting because they are single-message, so they can be used without violating the 2-message pattern of the protocol for computing on encrypted data. However, NIZK proofs require a setup phase, where a common reference string (CRS) is generated by a trusted third party, and handed to both the sender and the

---

6 In works prior to the appearance of [57], such cryptosystems are usually referred to as “homomorphic”, while homomorphic cryptosystems supporting both addition and multiplication are referred as “fully homomorphic.”
receiver. Thus, although the extra interaction needed for the setup can be amortized away by using (different portions of) one CRS for several executions of the protocol for computing on encrypted data, it does not strictly speaking achieve a 2-message protocol. Our approach requires no trusted setup. In a nutshell, we are able to make the receiver behave honestly, by making sure that the receiver learns the sender’s output only if its message is well formed, without the receiver having to prove it, or the sender having to explicitly verify it. We use techniques from [1, 59], which happen to work well with the additively homomorphic cryptosystem we utilize (and may not work for cryptosystems based on computational assumptions of a different flavor).
Chapter 2

Preliminaries

In this chapter, we present some definitions and notation used in this thesis.

2.1 General

We denote by $y \leftarrow A(x)$ the process of invoking the (possibly randomized) algorithm $A$ on input $x$ and assigning the result to $y$. We say that a function $\epsilon(k)$ is negligible if for every constant $c > 1$ we have $\epsilon(k) < 1/k^c$ for all sufficiently large $k$. We write $r \in_R S$ to indicate that $r$ is uniformly distributed on a finite set $S$, and $U_S$ to indicate the uniform distribution over $S$. We use the following standard notion of statistical distance:

**Definition 2.1.** [Statistical distance] Let $X, Y$ be random variables over the finite set $U$. Denote the distance between $X$ and $Y$ by

$$SD(X, Y) = \max_{U' \subseteq U} \left| \Pr_{x \leftarrow X} [x \in U'] - \Pr_{y \leftarrow Y} [y \in U'] \right|$$

We let POLY denote the set of functions computable in uniform polynomial time.

2.2 Randomized encodings of functions

**Encoding with randomized functions.** A key notion used throughout this thesis is that of a randomized encoding a function as implicit in [49], and explicit in [73, 74, 4].

**Definition 2.2.** [4] A function $g : X \times Y \to \{0, 1\}^*$, where $X = \{0, 1\}^\ell$, $Y = \{0, 1\}^{R(\ell)}$ is a perfectly (statistically, computationally) private randomized encoding of a function $f$ if it satisfies:

- Statistical Correctness: for all $x, x' \in X$, such that $f(x) \neq f(x')$, we have $SD(g(x, r), g(x', r)) \geq 0.5$. If there exists a “decoding” algorithm $D$, such that for all $x \in X$, we have $D(g(x, r)) = f(x)$

---

1To be precise, we in fact consider a family of functions, parameterized by $\ell$. 12
with probability 1 (equivalently, the statistical distance above is 1), we say the encoding is perfectly correct.

- Privacy: There exists a simulator algorithm $S$, such that for all $x \in X$, we have $S(1^{|x|}, f(x))$ is perfectly (statistically, computationally) indistinguishable from $g(x, U_Y)$.

Unless stated otherwise, correctness of encodings used in this thesis is assumed to be perfect. We say that an encoding is efficient, if there exists a polynomial in $|x|$ algorithm for computing $g$, and an efficient “decoder” algorithm $D$ as above.

**Randomizing polynomials (RP).** Our definition of randomizing polynomials is similar to the original definition from [73, Definition 2.1].

**Definition 2.3.** (Randomizing polynomials.) Let $r = (r_1, \ldots, r_m)$, and let

$$p(x_1, \ldots, x_n, r_1, \ldots, r_m) = (p_1(x, r), \ldots, p_s(x, r))$$

be a vector of polynomials over $F$. We say that $p$ is a perfectly (statistically, computationally) private RP for a function $f : F^n \to V$ if it satisfies the following requirements

- Privacy: as implied by Definition 2.2, taking $X = F^n, Y = F^m$.
- Statistical correctness: as implied by Definition 2.2 of statistical correctness taking $X = F^n, Y = F^m$.

We say that an RP encoding is of degree $d$, if each $p_i$ is of total degree at most $d$ when viewed as a polynomial in $(x, r)$. We will also consider the $r$-degree and $x$-degree separately, in which case variables of the other type do not count towards the degree.

### 2.3 Secure multi party computation

In this section, we outline some standard definitions of secure computation. These definitions are relevant primarily to the results in Chapter 3, and point out some specific features. Chapters dealing with other results, will either contain their own definitions, or point out differences, if any, relatively to the definitions below. Readers are referred to [26, 27, 60] for a more complete treatment.

**Communication model.** We consider the “plain” model, consisting of a network of $n$ processors, denoted $P_1, \ldots, P_n$ and referred to as parties. Each pair of parties is connected via a private, authenticated point-to-point channel. A more refined version, termed the client-server model, is defined and used in Section 3.3. In contrast to most previous work on the round complexity of MPC, we do not assume the availability of a broadcast channel, and require parties to only communicate via (synchronous) point-to-point channels.

---

2This is a certain generalization of the above definition of randomized encodings, allowing for non-boolean inputs. The case where the $x_i$’s are boolean is a useful special case, and can be viewed as encoding a partial function over $F^n$.

3For $F$ of characteristic other than 2, one cannot perfectly sample $Y$ uniformly using sequences of random bits, but it can be statistically sampled.
**Functionalities.** A secure computation task is defined by an $n$-party functionality, mapping the $n$ inputs (and, in the case of randomized functionality, additional randomness) to $n$ outputs. When we say “all functionalities” we refer by default to all polynomial-time computable functionalities.

Using standard reductions, we can consider without loss of generality deterministic, single-output functionalities which deliver their output to all $n$ parties. We refer to a functionality of this type as an $n$-party function $f : (\{0, 1\}^*)^n \rightarrow \{0, 1\}^*$. Realizing a general functionality $f'$ can be reduced (without additional interaction) to realizing a function $f$ as above by letting each input of $f$ include a share of the randomness for $f'$ (in case $f'$ is randomized) and defining the output of $f$ as the concatenation of the outputs of $f'$, where the $i$-th output of $f'$ is masked with a key contributed by party $P_i$.

**Protocol.** Initially, each party $P_i$ holds an input $x_i$, a random input $r_i$, and (in the case of computational or statistical security) a common security parameter $k$. The protocol proceeds in rounds, where in each round each party $P_i$ may send a “private” message to each party $P_j$. The messages $P_i$ sends in each round may depend on all its inputs ($x_i, r_i$ and $k$) and the messages it received in previous rounds. Finally, each party locally outputs some function of the messages it received, its input, and its private randomness. In the two-party case, the convention is that only one party speaks in each round, and the parties take turns speaking.

**Adversary.** We consider a malicious $t$-adversary $A$, where the parameter $t$ is referred to as the security threshold (or resilience). The adversary is an efficient interactive algorithm, who may choose a set $T$ of at most $t$ parties to corrupt. The adversary then starts interacting with a protocol (either a “real” protocol as above, or an ideal-process protocol to be defined below), where it takes control of all parties in $T$. In particular, it can read their inputs, random inputs, and received messages, and it can fully control the messages they send. We refer to such an adversary as malicious, and consider malicious adversaries unless stated otherwise.

**Security.** We use the standard UC security formulation [27]: a protocol is secure if for every real-world adversary $A$ there is a simulator $S$ such that no environment $Z$ can (interactively) distinguish between the real-world interaction of $A$ with the uncorrupted parties via the protocol and an ideal-world interaction of the simulator with the uncorrupted parties via the ideal functionality. In fact, we prove our results under a simplified version of UC security. Namely, as in the stand-alone model [60], we restrict the environment $Z$ to merely provide inputs and outputs to the parties, and do not allow it to otherwise interact with the adversary during the protocol execution. On the other hand, the simulator used in the proofs is straightline, in the sense that it does not “rewind” the adversary, and uses it as a black box. This allows to extend the security proofs of our protocols (again, from Chapter 3) to the full UC setting, assuming synchrony and authenticated channels are available (see [27, 87] for discussion on the subtleties of defining synchrony in the UC setting).

We distinguish between computational, statistical, and perfect security which are defined in a natural way by fixing the appropriate notion of indistinguishably. In this work, we consider two main notions of security: security with guaranteed output delivery (this is our default model), and a weaker notion called

---

4It is usually assumed that the adversary is given an “advice” string, or is alternatively modeled by nonuniform circuits.

5This corresponds to the non-adaptive security model, though our protocols in Chapter 3 are also secure against adaptive corruptions. The issue is less relevant to results in other chapters.
security with *selective abort*, introduced in [64]. In the former model, the ideal functionality will always deliver the correct outputs to all parties. Let us start by spelling out the precise security definition in the former model, in the presence of a static adversary, and explain how the definition is changed in the latter model.

Both *real-world* and *ideal-world* protocols, are executed in the presence of an adversary $A$, and an environment $Z$. The execution starts by $Z$ sending an input $x_i$ to party $P_i$, and sending to the adversary the set of parties $T$ it should corrupt. Also, for the cases of statistical and computational security, both the parties and $Z$ get a public security parameter $1^k$. In the *real-world* process, we assume by default that the adversary has a *rushing* capability: at any round it can first wait to hear all messages sent by uncorrupted parties to parties in $T$, and use these to determine its own messages. In the *ideal-world* process, we additionally have a “trusted party” realizing an idealized functionality $F$ which receives an input $x_i$ from every party $P_i$ (either corrupted or not), and returns $f(x)$ to each of the $P_i$’s. Indeed, each uncorrupted party simply forwards its input $x_i$ to $F$. The adversary $S$, also referred to as the simulator proceeds by invoking the adversary $A$ on randomness $r$ it generated for it, and the input $x_A$. It receives outgoing messages and feeds it with simulated incoming messages, until $A$ terminates its execution. At some point during the interaction, $S$ generates an input $x'_i$ for each corrupted $P_i$, and forwards it to $F$. After receiving all inputs, $F$ computes and forwards the output $f(x)$ to $S$, and to the uncorrupted parties. Note that $S$ may continue the interaction with $A$ after receiving out output from $F$ (in particular, $F$’s reply can be used for generating messages for the adversary).

We refer to such an adversary as a straight-line, black-box simulator. Upon the protocol’s termination (in both real and ideal protocols), the adversary sends $Z$ some function of its entire view, and every uncorrupted party sends $Z$ its output. We denote the concatenation of the adversary’s and the uncorrupted parties’ outputs in the real world by $Out_{A,Z}$, and in the ideal world by $Out_{S,Z}$. Finally, given the concatenated output $Out$, $Z$ outputs a single bit (intuitively, “guessing” whether it communicated with a real-world protocol, or with an ideal-world protocol).

We say that a protocol is perfectly (statistically) secure with straight-line simulation, if for any environment $Z$, and any real-world adversary $A$, there exists a straight-line simulator $S$ as above, such that the distributions $Z(Out_{S,Z})$, $Z(Out_{A,Z})$ are the same (have negligible in $k$ statistical distance). For computational security, we restrict $Z$ to be a polynomially bounded in overall input length and the security parameter $k$, and require the statistical distance between the above ensembles to be negligible.

In the latter model of security with selective abort, security (with straight-line simulation) is defined similarly, except that the functionality $F$ first delivers the outputs of corrupted parties to the simulator, then receives from the simulator a (possibly empty) set $G$ of indices, and then delivers to each uncorrupted $P_i$ with $i \in G$ its correct output and to each uncorrupted $P_i$ with $i \notin G$ a special abort symbol $\perp$. If the set of corrupted parties is empty, we require that $G$ is empty. We refer to this requirement as “non-triviality”. Intuitively, it is just a basic correctness requirement, implying that the protocol computes the alleged functionality in case no parties are corrupted (note that in the former model, an analogous notion is implied by the definition).

We also consider two common relaxations of both notions of security. Security in the *semi-honest model* is defined similarly to the above, except that the adversary (in both worlds) cannot modify the behavior of corrupted parties (only observe their secrets). *Privacy* is the same as above, except that the environment can only obtain outputs from the adversary (or simulator) and not from the uncorrupted parties. Intuitively, this only ensures that the adversary does not learn anything beyond what it could have learnt by submitting some
Another (less standard) variation of Privacy that we consider is privacy where the adversary also “knows” the outputs of the uncorrupted parties. More formally, the security notion is as for full security (in particular, the environment receives outputs from both the simulator and the uncorrupted parties), with the difference that the ideal functionality first delivers the corrupted parties’ output to the simulator, and then receives from the simulator an output to deliver to each of the uncorrupted parties. It sends each party the output received for it. If the adversary corrupts an empty set of parties, then it must be the case that all outputs equal $f(x)$ (that is, non-triviality). It does not otherwise guarantee, however, any form of correctness. We refer to this notion of privacy as “privacy with knowledge of outputs”. Observe that “regular” privacy does not imply privacy with knowledge of outputs. For instance, consider a private 2-round protocol for evaluating the xor of all inputs, where the output of each party depends on its Round 2 messages, and each party sends the same messages to all parties. Such a protocol indeed exists (the “basic” protocol from Section 3.5.1 is an example of such a protocol). Every such protocol is also secure with knowledge of outputs, since $S$ can run the uncorrupted parties’ reconstruction procedure on the Round 2 messages obtained by the simulation of $A$’s view, run $A$ to obtain the Round 2 messages sent to uncorrupted parties, and compute their outputs based on these messages. Now, augment the protocol, so that each party sends an additional bit $b$ to each of the other parties in Round 1. In Round 2, each party replies only to parties which sent $b = 1$ in Round 1. Clearly, this protocol remains private (simulate as before, except that if $b = 0$ was sent to some party in Round 1, they is no reply from that party). Consider a (real-world) adversary that sends $b = 0$ to all parties, and otherwise acts by the protocol. In this case, the adversary learns nothing, while the uncorrupted parties learn $f(x)$. Thus, the protocol is clearly not private with knowledge of outputs.

2.4 Representation models

This section is primarily relevant to the result in Chapter 5. Loosely speaking, a representation model is a way of interpreting strings as “programs” for evaluating (families of) functions over some finite domain. We are only interested in representation models which are universal in the sense that every function has a program evaluating it in that model. For simplicity we restrict the attention to functions defined over a binary input alphabet. An extension to the general case is straightforward.

Definition 2.4. [Representation model] A representation model is a polynomial-time computable function $U : \{0, 1\}^* \times \{0, 1\}^* \to \{0, 1\}^*$, where $U(P, x)$ is referred to as the value returned by a “program” $P$ on the input $x$. When $U$ is understood from the context, we use $P(x)$ to denote $U(P, x)$. We say that a function $f : \{0, 1\}^* \to \{0, 1\}^*$ can be implemented in a representation model $U$ if there exists an infinite sequence $(P_0, P_1, \ldots)$, referred to as an implementation of $f$ in $U$, such that $f(x) = U(P_{|x|}, x)$ for every $x \in \{0, 1\}^*$.

We now define the branching programs model. This is the representation model for which our main result applies.

Definition 2.5. [Branching program (BP)] A (deterministic) branching program over the variables $x = (x_1, \ldots, x_n)$ with input domain $I$ and output domain $O$ is defined by a tuple $(G = (V, E), v_0, T, \psi_V, \psi_E)$ where:
• $G$ is a directed acyclic graph.

• $v_0$ is an initial node of indegree 0. We assume without loss of generality that every $u \in V - \{v_0\}$ is reachable from $v_0$.

• $T \subseteq V$ is a set of terminal nodes of outdegree 0.

• $\psi_V : V \to [n] \cup O$ is a node labeling function assigning an output value to each terminal node in $T$, and a variable index from $[n]$ to each nonterminal node in $V - T$.

• $\psi_E : E \to 2^I$ is an edge labeling function, such that every edge is mapped to a non-empty set, and for every node $v$ the sets labeling its outgoing edges form a partition of $I$.

**BP evaluation.** The output $P(x)$ of a branching program $P$ on an input assignment $x \in I^n$ is naturally defined by following the path induced by $x$ from $v_0$ to a terminal node $v_\ell$, where the successor of node $v$ is the unique node $v'$ such that $x_{\psi_V(v)} \in \psi_E(v,v')$. The output is the value $\psi_V(v_\ell)$ labeling the terminal node reached by the path.

**BP complexity measures.** Let $P = (G(V,E),v_0,T,\psi_V,\psi_E)$ be a BP. The size of $P$ is $|E|$. (Note that in the case where $|I|$ is constant we have $|E| = O(|V|)$.) The height of a node $v \in V$, denoted height$(v)$, is the length (in edges) of the longest path from $v$ to a node in $T$. The length of $P$ is the height of $v_0$. We say that an implementation $(P_0,P_1,\ldots)$ of a function $f$ in the branching program model is length-bounded by $\ell(\cdot)$ if the length of each $P_n$ is at most $\ell(n)$.

**Remark 2.1.** In the following we will sometimes assume that branching programs have binary inputs and outputs, namely that $I = O = \{0,1\}$. We stress, however, that the generalization to non-binary domains is useful for some of the applications we have in mind. For instance, non-binary input alphabets are useful for casting the PIR protocol from [94] as a special case of our main construction, and large output alphabets are useful for applications such as private retrieval by keywords [31, 54].

Our protocols take the simplest form when the branching program being evaluated is layered in the following sense.

**Definition 2.6.** [Layered BP ] We say that $P$ is a layered branching program of length $\ell$ if the node set $V$ can be partitioned into $\ell + 1$ disjoint levels $V = \bigcup_{i=0}^{\ell} V_i$, such that $V_0 = \{v_0\}$, $V_\ell = T$, and for every $e = (u,v)$ we have $u \in V_i, v \in V_{i+1}$ for some $i$. We refer to $V_i$ as the $i$-th level of $P$.

Every branching program of size $s$ can be efficiently transformed into a layered branching program of size at most $s^2$ and the same length (cf. [113]). For convenience, we assume in our protocol that the server’s branching program is layered, which may square the server’s work but has no effect on the communication complexity or the client’s work. The quadratic overhead in the server’s work can be avoided in most useful special cases (e.g., evaluating decision trees or finite automata) and can be avoided in the general case if only client privacy is required.
2.5 Algebraic notation

We use \([n]\) to denote the set \([1, 2, \ldots, n]\). We let \(\mathbb{Z}_N\) denote the additive group modulo \(N\), and \(\mathbb{Z}_N^*\) denote the multiplicative group modulo \(N\). For a prime \(p\) and positive integer \(\ell\), we denote by \(\mathbb{F}_{p^\ell}\) the finite field of size \(p^\ell\) and characteristic \(p\). For \(\mathbb{F} = \mathbb{F}_{p^\ell}\), the trace function of \(\mathbb{F}\), \(\text{Tr}_\mathbb{F} : \mathbb{F} \rightarrow \mathbb{F}_p\), is defined by \(\text{Tr}_\mathbb{F}(a) = \sum_{i=0}^{\ell-1} a^i\) (see [97]).

We say a matrix \(M \in \mathbb{F}^{n \times n}\) is alternate, if \(M_{i,j} = -M_{j,i}\) for all \(i, j \in [n]\), and \(M_{i,i} = 0\) for all \(i \in [n]\). For matrices \(M \in \mathbb{F}^{h \times m}\) and \(N \in \mathbb{F}^{h \times n}\), we let \((M|N)\) denote the matrix resulting from concatenating \(N\) to the right of \(M\); similarly, for \(M \in \mathbb{F}^{m \times h}\) and \(N \in \mathbb{F}^{n \times h}\), we denote by \((M; N)\) the concatenation of \(N\) below \(M\). For subsets \(I \subseteq [h], J \subseteq [m]\), we let \(M_{I,J}\) denote the submatrix of \(M\) obtained by choosing rows \(I\) and columns \(J\). We sometimes abbreviate \(M_{I,[m]}\) as \(M_I\), \(M^{(i)}\) denotes \(M\)’s \(i\)'th column, and \(M_{(i)}\) denotes its \(i\)'th row. We let \(\text{colSpan}(M)\) (\(\text{rowSpan}(M)\)) denote the subspace spanned by \(M\)’s columns (rows). For square matrices \(M_1, \ldots, M_t\), the matrix \(\text{blocks}(M_1, \ldots, M_t)\) represents a block-diagonal matrix comprised of the blocks \(M_1, \ldots, M_t\) in that order. We denote by \(\{e_1, \ldots, e_n\}\) the standard basis of \(\mathbb{F}^n\). Vectors are by default column vectors, we use \(v^T\) to denote the row vector corresponding to column vector \(v\).
Chapter 3

On 2-round general MPC

In this chapter, we present our results on general 2-round MPC with honest majority.

3.1 Introduction

This thesis continues the study of the round complexity of secure multiparty computation (MPC) [124, 61, 14, 28]. Consider the following motivating scenario. Two or more employees wish to take a vote on some sensitive issue and let their manager only learn whether a majority of the employees voted “yes”. Given an external trusted server, we have the following minimalist protocol: each employee sends her vote to the server, who computes the result and sends it to the manager.

When no single server can be completely trusted, one can employ an MPC protocol involving the employees, the manager, and (possibly) additional servers. A practical disadvantage of MPC protocols from the literature that offer security against malicious parties is that they involve a substantial amount of interaction. This interaction includes 3 or more communication rounds, of which at least one requires broadcast messages.

The question we consider is whether it is possible to obtain protocols with only two rounds of interaction, which resemble the minimal interaction pattern of the centralized trusted server solution described above. That is, we would like to employ several untrusted servers instead of a single trusted server, but still require each employee to only send a single message to each server and each server to only send a single message to the manager. (All messages are sent over secure point-to-point channels, without relying on a broadcast channel or any further setup assumptions.)

In a more standard MPC setting, where there are $n$ parties who may contribute inputs and expect to receive outputs, the corresponding goal is to obtain MPC protocols which involve only two rounds of point-to-point communication between the parties.

The above goal may seem too ambitious. In particular:

- Broadcast is a special case of general MPC, and implementing broadcast over secure point-to-point channels generally requires more than two rounds [52].

- Even if a free use of broadcast messages is allowed in each round, it is known that three or more communication rounds are necessary for general MPC protocols which tolerate $t \geq 2$ corrupted
parties and guarantee output delivery, regardless of the total number of parties [56].

However, neither of the above limitations rules out the possibility of realizing our goal in the case of a single corrupted party, even when the protocols should guarantee output delivery (and in particular fairness). This gives rise to the following question:

**Question 3.1.** Are there general MPC protocols (i.e., ones that apply to general functionalities with \( n \) inputs and \( n \) outputs) that resist a single malicious party, guarantee output delivery, and require only two rounds of communication over point-to-point channels?

The above question may be highly relevant to real-world situations where the number of parties is small and the existence of two or more corrupted parties is unlikely.

Another possibility left open by the above negative results is to tolerate \( t > 1 \) malicious parties by settling for a weaker notion of security against malicious parties. A common relaxation is to allow the adversary who controls the malicious parties to *abort* the protocol. There are several flavors of “security with abort.” The standard notion from the literature (cf. [60]) allows the adversary to first learn the output, and then decide whether to (1) have the correct outputs delivered to the uncorrupted parties, or (2) abort the protocol and have *all* uncorrupted parties output a special abort symbol “⊥”.

Unfortunately, the latter notion of security is not liberal enough to get around the first negative result. But it turns out that a further relaxation of this notion, which we refer to as security with *selective abort*, is not ruled out by either of the above negative results. This notion, introduced in [64], differs from the standard notion of security with abort in that it allows the adversary (after learning its own outputs) to individually decide for each uncorrupted party whether this party will obtain its correct output or will output “⊥”. Indeed, it was shown in [64] that two rounds of communication over point-to-point channels are sufficient to realize *broadcast* under this notion, with an arbitrary number of corrupted parties. This gives rise to the following question:

**Question 3.2.** Are there general MPC protocols that require only two rounds of communication over point-to-point channels and provide security with *selective abort* against \( t > 1 \) malicious parties?

We note that both of the above questions are open even if broadcast messages are allowed in each of the two rounds.

### 3.1.1 Our results

We answer both questions affirmatively, complementing the negative results in this area with matching positive results.

- Our first main result answers the first question by showing that if only *one* party can be corrupted, then \( n \geq 5 \) parties can securely compute any function of their inputs with guaranteed output delivery by using only *two* rounds of interaction over secure point-to-point channels (without broadcast or any additional setup).

\[ ^{1}\text{We note that the negative result from [56] can be extended to rule out the possibility of achieving fairness in our setting with } t > 1. \]
We also prove a similar result in the client-server setting (described in the initial motivating example), where there are \( m \geq 2 \) clients who hold inputs and/or should receive outputs, and \( n \) additional servers with no inputs and outputs. For this setting, we obtain a general MPC protocol which requires a single message from each client to each server, followed by a single message from each server to each client. The protocol is secure against a single corrupted client and against coalitions of \( t < n/3 \) corrupted servers,\(^2\) and guarantees output delivery to the clients. We note that the proofs of the negative results from [56] apply to this setting as well, ruling out protocols that resist a coalition of a client and a server.

Our second main result answers the second question, showing that by settling for security with selective abort, one can tolerate a constant fraction of corrupted parties:

- There is a general 2-round MPC protocol over secure point-to-point channels which is secure with selective abort against \( t < n/3 \) malicious parties.

We note that the bound \( t < n/3 \) matches the security threshold of the best known 2-round protocols in the semi-honest model [14, 11, 73]. Thus, the above result provides security against malicious parties without any loss in round complexity or resilience. In the case of security against malicious parties, previous constant-round MPC protocols (e.g., the ones from [11, 55, 85]) require at least 3 rounds using broadcast, or at least 4 rounds over point-to-point channels using a 2-round implementation of broadcast with selective abort [64].

As is typically the case for protocols in the setting of an honest majority, the above protocols are in fact UC-secure [27, 93], and can provide statistical security for functionalities in low complexity classes such as NC\(^1\). Moreover, similarly to the constant-round protocols from [41, 100] (and in contrast to the protocol from [11]), the general version of the above protocols can provide computational security while making only a black-box use of a pseudorandom generator (PRG).\(^3\) This suggests that the protocols may be suitable for practical implementations.

Our results are motivated not only by the quantitative goal of minimizing the amount of interaction, but also by several qualitative advantages of 2-round protocols over protocols with three or more rounds. In a client-server setting, a 2-round protocol does not require servers to communicate with each other or even to know which other servers are employed. The minimal interaction pattern also allows to break the secure computation process into two non-interactive stages of input contribution and output delivery. These stages can be performed independently of each other in an asynchronous manner, allowing clients to go online only when their inputs change, and continue to (passively) receive periodic outputs while inputs of other parties may change. Finally, their minimal interaction pattern allows for a simpler and more direct security analysis than that of comparable protocols from the literature with security against malicious parties.

### 3.1.2 Related work

The round complexity of secure computation has been the subject of intense study. In the two-party setting, 2-round protocols (in different security models and under various setup assumptions) were given in [124, 21]. Achieving the latter threshold requires the complexity of the protocol to grow exponentially in the number of servers \( n \). When \( t = \Theta(n^{1/2} \log n) \), the complexity of the protocol can be made polynomial in \( n \).

\(^2\)This chapter also includes a simplified version of the second main result, based on the stronger assumption of the existence of a PRG in NC\(^1\), illustrating most of the ideas used in the stronger construction. We then show how to modify it.
Constant-round two-party protocols with security against malicious parties were given in [98, 88, 100, 81, 79, 75]. In [88] it was shown that the optimal round complexity for secure two-party computation without setup is 5 (where the model assumes that only one party can send messages in each round, and the negative result is restricted to protocols with black-box simulation).

More relevant to our work is previous work on the round complexity of MPC with an honest majority and guaranteed output delivery. In this setting, constant-round protocols were given in [6, 11, 10, 9, 73, 55, 36, 74, 38, 41, 85, 86, 30]. In particular, it was shown in [55] that 3 rounds are sufficient for general secure computation with $t = \Omega(n)$ malicious parties, where one of the rounds requires broadcast. Since broadcast in the presence of a single malicious party can be easily done in two rounds, this yields 4-round protocols in our setting. The question of minimizing the exact round complexity of MPC over point-to-point networks was explicitly considered in [85, 86]. In contrast to the present work, the focus of these works is on obtaining nearly optimal resilience.

Two-round protocols with guaranteed output delivery were given in [56] for specific functionalities, and for general functionalities in [38, 30]. However, the protocols from [38, 30] rely on broadcast as well as setup in the form of correlated randomness.

The round complexity of verifiable secret sharing (VSS) was studied in [55, 53, 86, 34]. Most relevant to the present work is the existence of a 1-round VSS protocol which tolerates a single corrupted party [55]. However, it is not clear how to use this VSS protocol for the construction of two-round MPC protocols. The recent work on the round complexity of statistical VSS [34] is also of relevance to our work. In the case where $n = 4$ and $t = 1$, this work gives a VSS protocol in which both the sharing phase and the reconstruction phase require two rounds. Assuming that two rounds of reconstruction are indeed necessary (which is left open by [34]), the number of parties in the statistical variant of our first main result is optimal. (Indeed, 4-party VSS with a single round of reconstruction reduces to 4-party MPC of a linear function, which is in $\mathsf{NC}^1$.)

Finally, a non-interactive model for secure computation, referred to as the private simultaneous messages (PSM) model, was suggested in [49] and further studied in [72]. In this model, two or more parties hold inputs as well as a shared secret random string. The parties privately communicate to an external referee some predetermined function of their inputs by simultaneously sending messages to the referee. Protocols for the PSM model serve as central building blocks in our constructions. However, the model of [49] falls short of our goal in that it requires setup in the form of shared private randomness, it cannot deliver outputs to some of the parties, and does not guarantee output delivery in the presence of malicious parties.

**Organization.** Section 2.3 includes the model of secure computation used in this result. Section 3.2 includes some additional preliminaries specific to this chapter. Section 3.3 presents a 2-round protocol in the client-server model. Our first main result (a fully secure protocol for $t = 1$ and $n \geq 5$) is presented in Section 3.4 and our second main result (security with selective abort for $t < n/3$) in Section 3.5.

### 3.2 Definitions and tools

We refer the reader to Section 2.3 for definitions of general secure computation used in this chapter. Another, very minimalistic, model for secure computation we will need is described in Section 3.2.1.
3.2.1 The PSM model

A private simultaneous messages (PSM) protocol [49] is a non-interactive protocol involving m parties \( P_i \), who share a common random string \( r \), and an external referee who has no access to \( r \). In such a protocol, each party sends a single message to the referee based on its input \( x_i \) and \( r \). These \( m \) messages should allow the referee to compute some function of the inputs without revealing any additional information about the inputs. Formally, a PSM protocol for a function \( f : \{0, 1\}^{\ell \times m} \rightarrow \{0, 1\}^* \) is defined by a randomness length parameter \( R(\ell) \), \( m \) message algorithms \( A_1, \ldots, A_m \) and a reconstruction algorithm \( \text{Rec} \), used by the referee to obtain the output. A PSM protocol needs to be secure against a semi-honest adversary corrupting the referee. Spelling out the security requirement for this setting, we obtain the following requirements.

- **Correctness:** for every input length \( \ell \), all \( x_1, \ldots, x_m \in \{0, 1\}^\ell \), and all \( r \in \{0, 1\}^{R(\ell)} \), we have \( \text{Rec}(A_1(x_1, r), \ldots, A_m(x_m, r)) = f(x_1, \ldots, x_m) \).

- **Privacy:** there is a simulator \( S \) such that, for all \( x_1, \ldots, x_m \) of length \( \ell \), the distribution \( S(1^\ell, f(x_1, \ldots, x_m)) \) is indistinguishable from \( (A_1(x_1, r), \ldots, A_m(x_m, r)) \).

We consider either perfect or computational privacy, depending on the notion of indistinguishability. (For simplicity, we use the input length \( \ell \) also as security parameter, as in [60]; this is without loss of generality, by padding inputs to the required length.)

A robust PSM protocol should additionally guarantee security with abort against a malicious adversary corrupting any subset of the parties \( P_1, \ldots, P_n \). That is, the effect of the messages sent by corrupted parties on the output can be simulated by either inputting to \( f \) a valid set of inputs (independently of the uncorrupted parties’ inputs) or by making the referee abort. Spelling out, the following requirement captures statistical robustness in this model.

- **Statistical robustness:** For any subset \( T \subseteq [m] \), there is an efficient (black-box) simulator \( S \) which, given access to the common \( r \) and to the messages sent by parties \( P_i \) for \( i \in T \), can generate a distribution \( x_T^* \) over \( x_i, i \in T \), such that the output of \( \text{Rec} \) on inputs \( A_T(x_T^*, r), A_{\bar{T}}(x_{\bar{T}}, r) \) is statistically close to the “real-world” output of \( \text{Rec} \) when receiving messages from the \( m \) parties, where the distribution is over random choice of \( r \) (note that \( r \) is the same in both real and ideal executions). In this definition, we allow \( S \) to produce a special symbol \( \perp \) (indicating “abort”) on behalf of some party \( P_i, i \in T \), in which case \( \text{Rec} \) outputs \( \perp \) as well.

The following theorem summarizes some known facts about PSM protocols that are relevant to our work.

**Theorem 3.1.** [49] (i) For any \( f \in \text{NC}^1 \), there is a polynomial-time, perfectly private and statistically robust PSM protocol. (ii) For any polynomial-time computable \( f \), there is a polynomial-time, computationally private and statistically robust PSM protocol which uses any pseudorandom generator as a black box.

For self-containment, the following section contains a full proof of the robust variants in Theorem 3.1, which are only sketched in [49, Appendix C].

3.2.2 Proof of Theorem 3.1

In Section 3.2.2.1, we prove Theorem 3.1 by a modular construction of robust PSM using several intermediate PSM constructions with certain properties. In that section, we either state the existence or construct
each of the intermediate PSMs. We formulate lemmas arguing their required properties, and show how to prove the theorem based on the lemmas. The lemmas are proved in subsequent sections.

3.2.2.1 A high level proof

We focus on boolean functions \( f : \{0, 1\}^m \rightarrow \{0, 1\} \). This is wlog., as a (robust) PSM can be trivially transformed to compute functions with a multiple-bit output, by executing it for each bit in parallel. To construct a statistically robust PSM, we employ a decomposable PSM protocol for a related function \( f' \). We say that a PSM is decomposable, if for each \( i \in [m] \), every output bit of \( A(x_i, r) \) depends on at most one bit of \( x_i \) (and depends arbitrarily on \( r \)).

Lemma 3.1. For any \( f \in NC^1 \), there is a polynomial-time, perfectly-private PSM protocol. This PSM is decomposable.

The proof of the above lemma is implied by combining [49, 72]. In particular, the transformation from PSM to statistically robust PSM is described in [49], but without a formal proof, and only for the case of \( m = 2 \) (PSM was initially defined in [49], who only considered \( m = 2 \)). Its proof requires the bulk of technical work in this section.

Lemma 3.2. For any \( f \in \text{POLY} \), there is a polynomial-time, computationally-private robust PSM protocol which uses a PRG as a black box. This PSM is decomposable.

Transforming decomposable PSM into statistically robust PSM. We use the following encoding scheme:

Construction 3.1. [a randomized encoding of strings] Given a length parameter \( \ell \), we define a randomized encoding of bits \( B^\ell_1 : \{0, 1\} \times \{0, 1\}^\ell \rightarrow \{0, 1\}^{2\ell} \) as \( B_1(b, r) = (a_0^b, a_1^b), \ldots, (a_0^b, a_1^b), \ldots \), where \( a_0^b = u + b \cdot d \), where the arithmetic is over \( \mathbb{F}_2 \). Extended to strings, we define (a family of) string encodings \( B((v_1, \ldots, v_h), (r_1, \ldots, r_h)) = B_1(v_1, r_1) \circ B_1(v_2, r_2) \circ \ldots B_1(v_h, r_h) \) (that is, simply output the concatenation of the encodings of the bits, using fresh randomness for each portion). Given an encoding \( a \in B_1(b) \), and \( c \in \{0, 1\}^\ell \), we define the ‘finger print’ of \( a \) under \( c \) as \( \text{fp}_c(a) = a_0^c, \ldots, a_\ell^c \). Similarly, extending to string encodings \( a_1, \ldots, a_h \in \{0, 1\}^\ell \) we define \( \text{fp}_c((a_1, \ldots, a_h), (c_1, \ldots, c_h)) = (\text{fp}_{c_1}(a_1), \ldots, \text{fp}_{c_h}(a_h)) \). For an encoding \( a \in \{0, 1\}^\ell \), we denote the portion of \( a \) encoding a bit \( v_i \) by \( a_i \).

First we provide a reduction from robust PSM to “somewhat robust PSM”. We define somewhat robust PSM for \( f : \{0, 1\}^m \rightarrow \{0, 1\} \) as a PSM \( \Pi'_f = (\ell, R'(\ell), A'_1, \ldots, A'_m, \text{Rec}) \) obtained form a decomposable PSM \( \Pi_f = (\ell, R(\ell), A_1, \ldots, A_m, \text{Rec}) \) for \( f \) via the construction below.

Construction 3.2. [SRPSM (somewhat robust PSM)]

- Let us write \( M_i(x_i, r) = A_i(x_i, r) = (M_{i,1}, \ldots, M_{i,\ell}) \), so that the string \( M_{i,j} \) depends only on \( x_{i,j} \) (the \( j \)-th bit of \( x_i \))^5. For convenience, we assume wlog. that all \( M_{i,j} \)'s are of the same length \( h(\ell) \).

\(^4\)For simplicity, we omit \( f \) here and wherever it is clear from the context.

\(^5\)The partitioning is fixed, assigning output bits independent of any \( x_j \) to \( M_{i,1} \).
• $A'_i(x_i, r), i \in [m]$:
  - Generate a random string $c_i \in \{0,1\}^{\ell \cdot h_1}$. For each $l \neq i \in [m]$, generate a sequence of random bits $s_{i,l,1}, \ldots, s_{i,l,\ell}$. All these are generated independently at random, using private coins.
  - Also, for each $j \in [\ell], l \in [m]$, there are two valid values $V_{i,j}^0, V_{i,j}^1$ of $M_{i,j}$, corresponding to $x_{i,j} = 0$ and $x_{i,j} = 1$ respectively (observe that these strings can be computed by all parties $P_i$, using the common randomness $r$). For each $l \neq i \in [m], j \in [\ell]$:
    * For each $b \in \{0,1\}$, let $a^b_{i,l,j} = B(V_{i,j}^b, r_{i,l,j})$, where $r_{i,l,j}$ is a corresponding (disjoint, designated for this computation) portion of $r$, the common randomness.
    * Let $out_{i,l,j} = B(V_{i,j}^{x_{i,j}}, r_{i,l,j})$.
  - Output the following sequence:
    * The string $c_i$.
    * For each $l \neq i \in [m], j \in [\ell], l \in [m]$, output $out_{l,i,j}$. (intuitively, this portion encodes $P_i$’s output, to be authenticated by others).
    * For each $l \neq i \in [m], j \in [\ell], l \in [m]$, let $p^i_{l,i,j} = f_c(a^b_{i,l,j})$. Output $veri_{l,i} = (p^i_{l,i,1}, p^i_{l,i,2}, \ldots, p^i_{l,i,\ell})$ (intuitively, this portion authenticates other parties’ output).
  - $Rec'(M_1, \ldots, M_m)$: For each $i \neq l \in [m], j \in [\ell], l \in [m]$, check that $f_{c_l}(out'_{l,i,j})$ equals either $p^0_{l,i,j}$ or $p^1_{l,i,j}$. If this is not the case for some $i, l, g$, output $\bot$. Otherwise, for each $i \in [m]$ decode each $out_i = (out_{i,1}, \ldots, out_{i,\ell})$ to obtain $out'_i$, and output $Rec(out'_1, \ldots, out'_m)$.

Next, we reduce statistically robust PSM to SRPSM ("somewhat robust PSM"). For

$$f(x_1, \ldots, x_m) : (\{0,1\}^\ell)^m \rightarrow \{0,1\},$$

define

$$f'((x_1^1, \ldots, x_1^\ell), \ldots, (x_m^1, \ldots, x_m^\ell)) : (\{0,1\}^{\ell \cdot m})^m \rightarrow (0,1) = f(x_1^1 + \ldots + x_1^\ell, \ldots, x_m^1 + \ldots + x_m^\ell),$$

where addition is vector addition over $\mathbb{F}_2$.

**Construction 3.3**

* Let $(A_1, \ldots, A_m, Rec, \ell)$ be an SRPSM for $f'$.

* $A'_i(x_i, r)$: Generate random strings $x_1^i, \ldots, x_\ell_i \in \{0,1\}^\ell$, so that $\sum_j x_j^i = x_i$ using private coins (for instance, for $j < \ell$ pick $x_j^i$ independently at random, and let $x_\ell^i = x_i - \sum_{j<\ell} x_j^i$). Output $A_i(x_1^i, \ldots, x_\ell_i, r)$ (using the joint randomness $r$). It will be convenient to denote the input of $A_i$ by $y_i = x_1^i, \ldots, x_\ell_i$ (a 'flat' vector of length $\ell^2$)

* $Rec'(M_1, \ldots, M_m)$: Output $Rec(M_1, \ldots, M_m)$.

We have that:

**Lemma 3.3.** Consider $f : \{0,1\}^{\ell \cdot m} \rightarrow \{0,1\}$. Then, construction 3.3 is a statistically robust PSM for $f$. Its privacy level is the same as that of the underlying SRPSM.

---

6Observe that by the choice of $y$, the output of $Rec'$ equals $f'(y) = f(x)$
Deriving Theorem 3.1. First consider a function \( f \in \text{NC}^1 \). By Construction, \( f' \) is also in \( \text{NC}^1 \) (each input bit is replaced by the sum (over \( \mathbb{F}_2 \)) of \( f \) input bits, which can be implemented by a depth-\( O(\log(\ell)) \) circuit). By Lemma 3.1, there exists a perfectly private, decomposable PSM for it. By Lemma 3.3, we conclude that there is a perfectly private, statistically robust PSM for \( f \). Now, assuming \( f \in \text{POLY} \), \( f' \) is clearly in POLY, and the existence of a polynomial-time, computationally private, statistically robust PSM for \( f \) follows by combining Lemma 3.2 with Lemma 3.3.

### 3.2.2.2 Proofs of Lemma 3.1, Lemma 3.2

We rely on (efficient) randomized encoding of functions as in Definition 2.2, which is additionally decomposable. That is, for all \( x \in \{0, 1\}^* \), every bit of \( g(x, r) \) depends on at most one bit of \( x \) (and depends on \( r \) arbitrarily). Observe that a decomposable randomized encoding for \( f \) can be trivially transformed into a PSM protocol for \( f \) (with a similar notion of privacy) by letting each party compute the output bits that depend on the input bits it holds, and \( r \) is given to the \( P_i \)'s as their common randomness (In particular, the input bits of \( f \) may be arbitrarily partitioned among the \( m \) parties in the PSM protocol.) Furthermore, the resulting PSM is decomposable, and polynomial-time assuming the encoding is efficient. Therefore it is sufficient to construct such an encoding in each of the cases.

**Proof of Lemma 3.1, sketch.** The encoding from \cite{74} is as we need, except for the decomposability property. More precisely, the following lemma is implicit in \cite{74}:

**Lemma 3.4.** \cite{74} Let \( f : \{0, 1\}^\ell \rightarrow \{0, 1\} \) denote a function in \( \text{NC}^1 \). Then there exists a polynomial \( h(\ell) \), and a perfectly private encoding of \( f \) by a vector of degree-3 randomizing polynomials over \( \mathbb{F}_2 \), such that all monomials are of degree at most 1 in the \( x_i \)'s, and at most 2 in the \( r_i \)'s.

We employ a simple idea from \cite{72} to transform this encoding into a decomposable one. Consider the encoding \( g(x, r), D(\cdot) \) as guaranteed by the lemma. We transform it into a new encoding \((g', D')\) via degree-3 randomizing polynomials:

- **g'(x, (r, z))**: Consider \( g(x, r) = (p_1(x, r), \ldots, p_h(x, r)) \). Recall that for each \( i \in [h(\ell)] \), \( p_i(x, r) = \sum_{j \in [\ell]} x_j \cdot q_{i,j}(r) + q_{i,0}(r) \), where the \( q_{i,j}(r)'s \) are of degree 2. Let \( z = (z_1, \ldots, z_h) \), where for each \( i \), \( z_i = (z_{i,0}, z_{i,1}, \ldots, z_{i,\ell}) \), such that \( z_i \) is a random solution to \( y_1 + \ldots + y_\ell = 0 \). Output the following polynomials:
  - For each \( i \in [h], j \in [\ell] \), a polynomial \( Q_{i,j} = x_j \cdot q_{i,j}(r) + z_{i,j} \).
  - For each \( i \in [h] \), the polynomial \( Q_{i,0} = q_{i,0}(r) + z_{i,0} \).

- **D'(Q, x)**: For each \( i \in [h] \), let \( p'_i = \sum_{j \in [\ell]} Q_{i,j} + Q_{i,0} \). Output \( D(p'_1, \ldots, p'_h) \).

Clearly, the encoding is correct, since for each \( i \in [\ell] \), \( \sum_{j \in [0, \ldots, \ell]} Q_{i,j}(x, r, z) = p_i(x, r) \). Intuitively, privacy follows from the fact that \((Q_{1,1}, \ldots, Q_{i,\ell})\) only reveals \( \sum_{j \in [0, \ldots, \ell]} Q_{i,j} = p_i(x, r) \) (due to the \( z_{i,j} \)'s). In turn \((p_1(x, r), \ldots, p_\ell(x, r))\) depend only on \( f(x) \). 

\[ \square \]
Proof of Lemma 3.2, sketch. Yoga’s construction can be viewed as an (efficient) randomized encoding scheme for functions \( f(x_1, \ldots, x_\ell) \in \text{POLY} \) (as implicit in [4]). That is, there exist (efficient) PPT algorithms \((\text{Enc}, \text{Dec})\) such that:

- Given a circuit \( C \) with \( \ell \) input bits, \( \text{Enc}(C) \) outputs a string \( C' \) consisting of \( \ell \) possibly correlated pairs of keys \( (W_i^0(x), W_i^1(x)) \). \( \text{Enc} \) uses a PRG in a black-box manner (any PRG would do). We denote \( \text{Enc}(C)_x = W_1^x, \ldots, W_\ell^x \).
- Correctness: For every \( C \) and \( x \), we have \( \text{Dec}(\text{Enc}(C)_x) = C(x) \).
- Privacy: There is a PPT \( S \) such that for all \( C \) and \( x \), it holds that \( S(|C|, C(x)) \) and \( \text{Enc}(C)_x \) are computationally indistinguishable.

The above scheme induces an efficient decomposable randomized encoding for any \( f \in \text{POLY} \) by fixing a (poly-sized) circuit \( C \) implementing it, and letting \( g(x, r) = \text{Enc}(C, r)_x \), and \( D(O) = \text{Dec}(O) \). In particular, decomposability follows from the fact that every output bit of \( \text{Enc}(C)_x \) depends on at most one bit of some \( x_i \) (has degree 1 in \( x_i \)).

3.2.2.3 Proof of Lemma 3.3

Let \( \Pi''_f \) denote the PSM protocol for \( f \) from Construction 3.3. Let \( \Pi'_f \) denote the SRPSM it employs, and let \( \Pi_f \) denote the (decomposable) PSM protocol the SRPSM employs. We prove that \( \Pi''_f \) is a PSM as deduced in Lemma 3.3.

Correctness proof. Follows immediately from the correctness of \( \Pi_f \), and the construction.

Privacy. It is sufficient to prove that \( \Pi'_f \) preserves the privacy notion of \( \Pi_f \). This holds since the output of \( f' \) equals the output of \( f \), so given \( f(x) \), one can simulate \( \text{Rec}' \)'s input distribution simply by feeding \( f'(y) = f(x) \) to the simulator guaranteed to exist for \( \Pi'_f \), proving \( \Pi''_f \) is secure with the same privacy notion as \( \Pi'_f \). So, we prove that given a PSM \( \Pi_g \) for some function \( g : \{0,1\}^\ell \to \{0,1\} \), its transformation into SRPSM results in a PSM \( \Pi'_g \) having the same notion of privacy. A corresponding simulator \( S'(1^{\ell}, \text{out} = g(x)) \) operates as follows:

- Execute the simulator \( S(1^{\ell}, \text{out}) \) to obtain the sequence \( M_1^*, \ldots, M_m^* \).
- For each \( l \in [m] \), let \( c_l \) be a random string in \( \{0,1\}^{\ell h(l)} \).
- For each pair \( l \neq i \in [m], j \in [\ell] \), let \( \text{out}_{i,l}^* \) be a random independent encoding of \( M_{i,j}^* \).
- For \( i \neq l \in [m], j \in [\ell] \) let \( s_{i,l,j} \in \{0,1\} \) be a random and independent bit. Set \( p_{i,l,j}^{s_{i,l,j}} = f_{c_l}(\text{out}_{i,l,j}^*) \)
  and \( p_{i,l,j}^{1-s_{i,l,j}} \) a random independent value in \( \{0,1\}^{\ell h(l)} \). Generate \( \text{verif}_{i,l} \) based on these values as in Construction 3.2.

\(^7\) Observe that \( \text{Enc}(C)_x \) can be expressed as a degree-2 polynomial in the bits of \( W_i^b \)'s and \( x_i \)'s over \( \mathbb{F}_2 \). In particular, we treat the bits which depend only on the randomness at part of \( W_i^{x_i} \).
First observe that for a perfectly private \( \Pi_g \), and given a perfect simulator \( S \) demonstrating it, \( S' \) is a perfect simulator of the referee’s view\(^\star \). To see that, we rely on the following simple observation:

**Observation 3.1.** For any \( c \) (of the proper length), and a random encoding \( a \) of a string \( v \in \{0, 1\}^b \), it holds that \( \text{fp}_c(a) \) is independent of the value of \( v \).

**Proof.** For \( v \in \{0, 1\} \), it holds that \( \text{fp}_c(B(0, \cdot)) \) and \( \text{fp}_c(B(1, \cdot)) \) are distributed as random strings. Since

\[
\text{fp}_c(a) = \text{fp}_c(a_1) \circ \ldots \circ \text{fp}_c(a_{|v|})
\]

and each \( a_i \) is a random independent encoding of some bit, it follows that \( \text{fp}_c(a) \) is a random string of length \( \ell \cdot |v| \) (and is therefore independent of \( v \)).

Now, assume the privacy of \( \Pi_g \) is computational. Observe that \( S'(\ell, F(x)) \) is distributed as \( O_{\text{sim}} = M' \leftarrow S(\ell, F(x)), Y'(M') \), where \( Y' \) is a randomized function where fresh (independent) randomness is used both \( S \), \( Y' \) (the randomness is implicit in the notation). Also, the output of \( \Pi_g' \) is of the form

\[
O_{\text{real}} = M \leftarrow \Pi_g(x), Y(M)
\]

using fresh random strings. Now, given an algorithm \( A \) with polynomial \( p^{-1}(\ell) \) distinguishing advantage between \( O_{\text{sim}} \) and \( O_{\text{real}} \) for an infinite number of \( x \)'s, we obtain a distinguisher \( A' \) for the same \( x \)'s and same advantage between \( S(\ell, g(x)) \) and \( \Pi_g(x) \). \( A'(M) \) computes \( M' \leftarrow Y'(M), \text{out} = A(M) \) and outputs \text{out}. In case \( M \) comes from the simulator, \( M \) is distributed exactly as \( O_{\text{sim}} \). If \( M \) comes from \( \Pi_g \) by \(^\star \), \( M \) is distributed exactly as \( O_{\text{real}} \), so we obtain \( A \)'s distinguishing advantage for this input (contradicting the privacy of \( \Pi_g \)).

**(Statistical) Robustness, sketch.** Let \( A \) denote an unbounded (deterministic) adversary attacking a subset \( T \subset [n] \) in \( \Pi_g' \). A simulator \( S \) for \( A \) given \( \ell \), the common randomness \( r \) and access to \( A \) acts as follows:

1. For each \( i \in T \), invoke \( P_i \) to receive its output sequence \( M_i \) (given the common randomness \( r \)).

2. For each pair \( l \neq i \in T \), check whether \( \text{verif}_{l,i}^* \) is consistent with \( \text{out}_{l,i,j}^* \) for all \( j \in [\ell^2] \) (that is, not causing \( \text{Rec} \) to output \( \perp \)). If not, set \( x_j^* = \perp \) for all \( j \in T \), output \( x_T^* \), and halt.

3. Otherwise, for each \( i \in T, l \in [m] \sim T \):
   
   (a) If for some \( j \in [\ell^2] \), \( \text{out}_{l,i,j}^* \) does not equal \( B(V_{i,j}, r_{l,i,j}) \) or \( B(V_{i,j}, r_{l,i,j}) \), set \( x_j^* = \perp \) for all \( j \in T \), output \( x_T^* \), and halt.

   (b) For each \( j \in [\ell^2] \), compute \( a_{l,i,j}^b \) (as in the protocol, using \( r \)) for \( b \in \{0, 1\} \). Count the number \( n_{l,i,j} \) of such \( \text{fp}_{c_i}^* \)’s \((0, 1 \text{ or } 2)\) contained in the set \( \{p_{l,i,j}^0, p_{l,i,j}^1\} \), and let \( q_{l,i,j} = n_{l,i,j}/2 \). Set \( q_{l,i} = \Pi_{j \in [\ell^2]} q_{l,i,j} \). With probability \( 1 - q_{l,i} \) output \( x_i^* = \perp \) for each \( i \in T \), and halt.

4. For each \( i \in T, j \in [\ell^2] \), let \( y_{i,j} = b_{i,j}^* \) where \( \text{out}_{l,i,j}^* \in B(V_{i,j}^{b_{i,j}}, \cdot) \) for all \( l \neq i \in [m] \). For \( i \in T, z \in [\ell] \) set \( x_{i,z}^* = \sum_{j \in \{0, \ldots, \ell - 1\}} y_{i,j \cdot \ell + z} \).

We now sketch the correctness of this simulator. We prove that for all \( x_T^* \in (\{0, 1\}^\ell)^m \) and shared randomness \( r \), the distribution of \( O_{\text{sim}} = \text{Rec}(x_T^*, x_T^*) \) is at most \( 2^{1 - \ell} \)-far from \( O_{\text{real}} \), the output of \( \text{Rec} \) in the real-world protocol. In the latter, the probability is over the random choices of the uncorrupted parties. There are several possible cases.

- The checks in Item 2 do not pass. In this case, \( x_T^* = \perp \) for all \( i \in T \), and the output of \( \text{Rec} \) is \( \perp \). The output of \( \text{Rec} \) in the real execution is also \( \perp \) (with probability 1).
• Otherwise, the checks in Item 3a do not pass. That is, there exists \( i \in T, l \in [m] \setminus T, j \in [\ell^2], \) such that \( \text{out}_{i,l,j} \) belongs neither to \( B(V_{i,j}^0, r_{i,j}) \) nor to \( B(V_{i,j}^1, r_{i,j}) \). In this case, \( \text{Rec}(x_T^0, x_T^1) = \perp. \) In the real-world, \( \text{Rec} \) outputs \( \perp \) with probability at least \( 1 - 2^{-\ell}. \) This is because the authentication by \( P_l \), via \( \text{verif}_{i,l,j} \), fails with probability at least \( 1 - 2^{-\ell}. \) This, in turn, follows directly from the following observation.

**Observation 3.2.** Let \( a = B_1(b, r) \) for some \( b \in \{0, 1\}, r \in \{0, 1\}^\ell. \) Let \( c \in \{0, 1\}^\ell \) be a random string. Then, for any \( a' = B_1(b', r') \neq a, \) \( \Pr_c[fp_c(a) = fp_c(a')] \leq 2^{-\ell}. \)

**Proof.** If \( b = b' \), then \( fp_c(a) \neq fp_c(a') \) with probability 1 (by definition of the encoding, and the function \( fp_c(\cdot) \)). Otherwise, for each \( i \in [\ell] \), either \( a^0_i \neq a'^0_i \) or \( a^1_i \neq a'^1_i \), so the corresponding bits in \( fp_c(\cdot) \) do not match with probability \( 1/2 \) (over the choice of \( c \)). Since the bits in \( c \) are selected independently from each other, the probability of a match is \( 2^{-\ell}. \) \( \Box \)

• Otherwise, only the checks in Item 3b possibly fail. The simulator outputs a non-\( \perp \) sequence \( x_T^* \) (as calculated in Item 4) with probability \( q = \Pi_{i \in T, l \in [m] \setminus T} q_{i,l} \), as determined by the \( \text{out}_{i,l,j} \)'s. It outputs \( x_T^* = \perp \) for all \( i \in T \) with probability \( 1 - q. \) Consider the real-world execution for some input \( x_T \). In this case, \( \text{support}(O_{\text{sim}}) = \text{support}(O_{\text{real}}) \subseteq \{ \text{Rec}(x_T^*, x_T^1), \perp \}. \) This is so since we assume \( T \subset [n] \) (a proper subset), so there is at least one uncorrupted party \( P_l \) verifying the bad party’s outputs, and the checks in Item 3a do not merely pass because there are no uncorrupted parties. For a fixed \( j \in [\ell^2], \) \( \text{verif}_{i,l,j} \) reveals an inconsistency with \( a^m_{i,l,j} \) iff it’s not contained in \( \{ p_{i,l,j}^0, p_{i,l,j}^1 \}. \) Let \( \tilde{q}_{i,l,j} \) denote the probability (over the choice of \( y \)) that \( \text{verif}_{i,l,j} \) “disqualifies” \( P_l \) (that is, \( fp_c(a^m_{i,l,j}) \notin \{ p_{i,l,j}^0, p_{i,l,j}^1 \} \)), causing \( \text{Rec} \) output \( \perp. \) Clearly, the probability that \( \text{Rec} \) outputs a non-\( \perp \) value is \( \tilde{q} = \Pi_{i \in T, l \in [m] \setminus T, j \in [\ell^2]} \tilde{q}_{i,l,j}. \) There are several Options:

- If for some \( i \in T, l \in [m] \setminus T, j \in [\ell^2], q_{i,l,j} = 0, \) thus \( q = 0, \) and both \( O_{\text{real}} \) and \( O_{\text{sim}} \) are \( \perp \) with probability 1.

- Otherwise (for all \( i \in T, l \in [m] \setminus T, j \in [\ell^2], n_{i,l,j} \in \{1, 2\} \)). For \( z \in [\ell], \) let \( I_z = \{ j : j = z, j \in [\ell] \setminus T, z \in [\ell] \}. \) Fix some for some \( i \in T, l \in [m] \setminus T, z \in [\ell]. \) There are several options:

  * There is at least one \( n_{i,l,j} \) for \( j \in I_z \) which equals 2. In this case, \( \text{verif}_{i,l,j} \) “disqualifies” \( P_l \) for some \( j \in I_z \) with probability exactly \( \Pi_{j \in I_z} q_{i,l,j} \), since \( y_{g_j} \)'s for \( j \in I_z \) are random, and \( \ell - 1 \) wise independent.

  * Otherwise, the probability that \( \text{verif}_{i,l} \) disqualifies \( P_l \) due to some \( j \in I_z \) is \( 2^{1-\ell} \cdot w_0, \) where \( 2^{1-\ell} \) is the probability to avoid disqualification for the first \( \ell - 1 \) values in \( I_z \), and \( w_0 \) is either 0 or 1 (since \( y_{i,(\ell-1)} \cdot x_{l,z} \) is determined by the other \( \ell - 1 \) bits in the set, and \( x_{l,z} \)).

If for all \( i \in T, l \in [m] \setminus T, z \in [\ell] \) the second case never occurs, then the probability \( \tilde{q} \) of \( \text{Rec} \) outputting a non-\( \perp \) value is exactly \( q \) (as in the simulation). Otherwise, \( q \leq 2^{-\ell}, \) and \( \tilde{q} \leq 2^{1-\ell}, \) so the difference between them is at most \( 2^{1-\ell}, \) which also bounds the statistical distance between \( O_{\text{sim}} \) and \( O_{\text{real}}. \) \( \Box \)
3.2.3 Secret sharing schemes

This section includes standard definitions of secret sharing schemes, along with several less common or new properties of such schemes that we will need.

We start with recalling the (standard) notion of secret sharing. Let \( \text{Sec} \) be a domain of “secrets” to share, \( \mathcal{R} \) be a randomness domain, \( \mathcal{S} \) be a shares domain (all are finite), and \( A \subseteq 2^{[n]} \) an “access structure”. A \((n,A)\) secret sharing scheme is defined by a “dealing” function \( D : \text{Sec} \times \mathcal{R} \rightarrow \mathbb{S}^n \), and recovery functions \( \text{Rec}_A : \mathbb{S}^{|A|} \rightarrow \text{Sec} \), for each \( A \in \mathcal{A} \). We denote \( D(\text{sec}, r) = s = (s_1, \ldots, s_n) \) and \( s_A = (s_i)_{i \in A} \). The scheme satisfies:

- **Privacy.** For all \( \text{sec}, \text{sec}' \in \text{Sec} \), and all \( A \not\in A \), \( D(\text{sec}, \mathcal{R})_A \) and \( D(\text{sec}', \mathcal{R})_A \) are identically distributed.

- **Correctness.** For all \( \text{sec} \in \text{Sec}, r \in \mathcal{R} \) and \( A \in \mathcal{A} \), we have \( \text{Rec}_A(D(\text{sec}, r)_A) = \text{sec} \).

We refer to a scheme where \( A = \{ A \subseteq [n] | |A| > t \} \) as a \((n, t)\) threshold secret sharing scheme. In this work, we use only threshold secret sharing schemes. For most known secret sharing schemes (CNF, Shamir, bi-variate Shamir etc.), given \( (D(s, r)_A, A) \) for some \( s \in \text{Sec}, r \in \mathcal{R} \), the algorithm \( \text{Rec}_A \) is efficient. The only additional property we often require, is the ability to efficiently check whether a purported partial sharing to a qualified set \( A \in \mathcal{A} \) is consistent with some secret. Unlike the typical use of secret sharing in the context of MPC, (most of) our constructions do not rely on linearity or multiplicativity (to be defined) of the secret sharing scheme.

One property of secret sharing scheme we will need is efficient extendability. A secret sharing scheme is efficiently extendable, if for any subset \( T \subseteq [n] \), it is possible to efficiently check whether the (purported) shares to \( T \) are consistent with a valid sharing of some secret \( s \). Additionally, in case the shares are consistent, it is possible to efficiently sample a (full) sharing of some secret which is consistent with that partial sharing. This property is satisfied, in particular, by the schemes mentioned above, as well as any so-called “linear” secret sharing scheme.

**Multiplicative secret sharing.** A secret sharing scheme is \( d\)-multiplicative, if given sharings of \( d \) secrets \( \text{sec}^1, \ldots, \text{sec}^d \), each party can locally compute an additive share of \( \prod_{j=1}^d \text{sec}^j \). More precisely, we define:

**Definition 3.1.** \([d\text{-Multiplicative secret sharing}]\) Consider a \((n,A)\) secret sharing scheme where \( \text{Sec} = \mathbb{F} \) for some finite field \( \mathbb{F} \). Let \( \text{sec}^1, \ldots, \text{sec}^d \in \text{Sec}, r^1, \ldots, r^d \in \mathcal{R} \), and let \( D(\text{sec}^j, r^j) = (s_1^j, \ldots, s_n^j) \) for each \( j \in [d] \). The scheme is \( d\)-multiplicative, if there exists a function \( \text{MULT} : [n] \times \mathbb{S}^d \rightarrow \mathbb{F} \), such that for all \( \{ \text{sec}^j, r^j \}_{j \in [d]} \) as above, \( \sum_{i=1}^n \text{MULT}(i, s_1^j, \ldots, s_n^j) = \prod_{j=1}^d \text{sec}^j \).

**Linear secret sharing.** We say that a secret sharing scheme is linear (an LSSS), if \( \text{Sec} = \mathbb{F}, \mathcal{R} = \mathbb{F}^m \) for some finite field \( \mathbb{F} \) and number \( m > 0 \), and each share \( g_i \) equals \( l_{i,1}(\text{sec}, r), \ldots, l_{i,n}(\text{sec}, r) \), where each function \( l_{i,j} \) is a fixed linear combination of \( s, r_1, \ldots, r_m \) over \( \mathbb{F} \). It turns out to be convenient to represent a linear scheme by a monotone span program (MSP) [83] for its access structure \( A \) represented as a (monotone) function \( f_A : 2^{[n]} \rightarrow \{0,1\} \) in the natural way. A span program is defined as follows.

**Definition 3.2.** A monotone span program (MSP) is a tuple \( \mathcal{M}(M, \text{tar}, \psi, \mathbb{F}) \), where \( \mathbb{F} \) is a finite field, \( M \) is a \( d \times e \) matrix over \( \mathbb{F} \), \( \text{tar} \) is a non-zero vector over \( \mathbb{F} \), \( \psi : [d] \rightarrow [n] \) is a “row labeling function, and \( \text{tar} \in \mathbb{F}^n \).
is a non-zero vector\(^8\). For a subset \(A \subseteq [n]\), denote by \(M_A\) the submatrix comprised from all rows \(j\), such that \(\psi(j) \in A\). We refer to a vector \(v \in \mathbb{F}^d\) for which \(v \cdot M = \text{tar}\), and only entries corresponding to \(M_A\) are non-0 as a combination vector for \(x\) (for convenience we often treat it as a vector in \(\mathbb{F}^{m_x}\), where \(m_x\) is the number of rows in \(M_A\)). The function \(f : \{0, 1\}^n \rightarrow \{0, 1\}\) computed by \(M\) is 1 on input \(x \in \{0, 1\}^n\) iff. there exists a combination vector for \(M_A\). We refer to \(d\) as the MSP size (the number of columns is, without loss of generality, not larger than the number of rows).

MSP’s and LSSS turn out to be equivalent in the sense that one can construct an LSSS for an access structure \(A\) from an MSP computing it and vice versa over the same field and with the same size [12, 83]. In particular, the transformations (in both directions) map each share \(s_{i,j}\) given to party \(i\) to a row labeled by \(i\) (the \(j\)’th among such rows). We will explicitly use the following transformation from MSP to LSSS. Given an MSP \(M(M, \text{tar}, \psi, \mathbb{F})\) computing a (monotone) function \(f_A\), an LSSS for \(A\) is defined as follows. The algorithm \(D\) on input \(sec \in \mathbb{F}\), picks a random solution \(r\) to \(\text{tar} \cdot r^T = s\), and gives party \(i\) the vector \(s_i = M_{i(1)} \cdot r^T\). Given a set \(A \in A\), let \(h_A\) denote a combination vector for \(M_A\) (if there are several, pick one arbitrarily). The algorithm \(\text{Rec}\) outputs \(h_A \cdot s\). In the following, we sometimes switch back and forth between the two representations without explicit mention.

**Pairwise verifiable secret sharing.** We put forward a simple verifiability notion of secret sharing schemes. Intuitively, a pairwise-verifiable secret sharing scheme is obtained by taking a secret sharing scheme \((D^E, \text{Rec}^E)\), and appending auxiliary information \(D^V\) to the sharing in a way that does not compromise privacy. A sharing \(s^E\) in is consistent with \(D^E(\text{sec}, r)\), if the augmented sharing \(s^E\) (along with auxiliary shares) given to parties within any \(A \in A\) pass all prescribed pairwise checks. More formally.

**Definition 3.3.** A \((n, A)\) secret sharing scheme \((D : \text{Sec} \times R \rightarrow S, \{\text{Rec}_A\}_{A \in A})\) is pairwise-verifiable if it satisfies:

- There exist algorithms \(D^E : \text{Sec} \times R \rightarrow S_1, D^V : \text{Sec} \times R \rightarrow S_2\), such that

  \[
  D(\text{sec}, r) = ((D^E(\text{sec}, r)_1, D^V(\text{sec}, r)_1), \ldots, (D^E(\text{sec}, r)_n, D^V(\text{sec}, r)_n)).
  \]

  \(\text{Rec}\) is independent of \(D^V\)’s output.

- Let \(\text{Rec}_A^E(s_1, \ldots, s_{|A|}) = \text{Rec}_A((s_1, 0), \ldots, (s_{|A|}, 0))\). Then, \((D^E, \text{Rec}^E)\) is a \((n, A)\)-secret sharing scheme, to which we refer as the “effective” scheme, and the output of \(D^E\) is referred as the “effective” shares.

- There exists a set of functions \(\{V_{i,j}(x), U_{i,j}(x)\}_{i<j<|n|}\) in POLY, such that the following holds. For any subset of parties \(A \in A\), and for any (purported) partial sharing \(s_A\), if \(V_{i,j}(s_i) = U_{i,j}(s_j)\) for all \(i < j \in A\), then \(s^E\) is consistent with a sharing \(D^E(\text{sec}, r)\) for some \(\text{sec} \in \text{Sec}, r \in R\).

A somewhat stronger (and more natural) definition would address secret sharing schemes \((D, E)\) without any special structure, and simply require that if all pairwise checks within some \(A \in A\) pass, then the sharing to \(A\) is consistent with some \(D(\text{sec}, r)\). However, we only manage to transform general LSSS schemes (it

\(^8\)It can be assumed, without loss of generality, that \(\text{tar} = (1, 0, \ldots, 0)\).
is convenient to rely on such schemes in our application) into pairwise-verifiable schemes satisfying this somewhat weaker definition, which is also sufficient for our application. We observe that the more natural definition can be achieved for LSSS schemes in which the rows held by a qualified set span the rows of the MSP, which is the case in many common LSSS’s such as Shamir, bi-variate Shamir and CNF.

We prove the following theorem implicit in [39], which might be of independent interest (when reformulated this way). A proof, also implicit in [39], is included for completeness.

**Theorem 3.2** Every $(n, A)$ LSSS $L'$ can be transformed into a pairwise-verifiable $(n, A)$ LSSS $L$ (over the same field), such that $L'$ is its effective scheme. In particular, if $L'$ is $d$-multiplicative, then so is $L$ (using the same function $\text{MULT}$, applied to $s'$). The share complexity of $L'$ is at most quadratic in the complexity of $L'$.

**Proof** Throughout the proof, we assume (only for notational simplicity) that $d = n$ and $\psi(i) = i$, the extension of the proof to the general case, to be explained later, is straightforward. Consider an MSP $\mathcal{M}$ specified by a $d \times e$ matrix $M$ over a field $F$ (and target vector $(1, 0, \ldots, 0)$) evaluating $f : \{0, 1\}^n \rightarrow \{0, 1\}$, which is equivalent to an LSSS $(D', E', A_f)$ (that is, the LSSS is obtained from the MSP as described above). For a matrix $B$, $B_i$ denotes $i$'th row, $\text{Span}(B)$ denotes the span of its rows. For sets $I, J$, denote by $B_{I \times J}$ the submatrix of $B$ restricted to these rows and columns.

We define the algorithm $D$ as follows. To distribute a secret $s$ pick a random symmetric $e \times e$ matrix $R$ under the restriction that $R_{0,0} = s$. Give party $i$ the share $s_i = R \cdot i$ (here and elsewhere, we sometimes neglect to explicitly transpose vectors in products with matrices). The value $s_{i,1}$ is interpreted as $s_i^E$, while the rest of the vector, $s_i[2, \ldots, e]$, is interpreted as $s_i^V$. For $i < j$, we define $V_{i,j}(s_i) = \langle M_i, s_j \rangle, U_{i,j}(s_j) = \langle M_j, s_i \rangle$ ($\langle \cdot, \cdot \rangle$ denotes a dot product). On input $(s_1, \ldots, s_{|A|})$, the algorithm $\text{Rec}_A$ outputs $\langle h_A, s_A^E \rangle$.

We start with proving that the resulting scheme is an $(n, A)$ secret sharing scheme. As to correctness, a set $A \in A$ recovers the secret since $s_i^E = s_{i,1} = R_1 \cdot i = M_i \cdot R^T_1$. Therefore, $\langle h_A, s_A^E \rangle = h_A \cdot M_A \cdot R^T_1 = (1, 0, \ldots, 0) \cdot R^T_1 = s$. As to privacy, we prove that a non-qualified set $T \notin A$ has no information about $s$. Let $M_A \cdot R = X$ denote the set of shares received by $A$ (indeed, $X_i = s_i$, as $R$ is symmetric). Let $R'$ be a matrix satisfying $M_A \cdot R' = X$. By correctness of the MSP, we have that $\text{tar} = (1, 0, \ldots, 0) \notin \text{Span}(M_A)$. We observe that exists a vector $\mu = (\mu_1, \ldots, \mu_e)$ with $\mu_1 = 1$ so that $M_A \cdot \mu = 0$. To see this, perform Gaussian elimination on $M_A$, such that the last non-zero coordinate in row $i$ is larger than the last non-zero coordinate in row $i + 1$. Denote the resulting matrix by $M'_A = B \cdot M_A$, where $B$ is a non-singular $d \times d$ matrix representing the elimination. Since $\text{tar} \notin \text{Span}(M_A)$, none of the rows has a non-zero value only at the first coordinate. Therefore, we manage to construct a solution $\mu$ to $M'_A \cdot \mu$ with $\mu_1 = 1$ proceeding from the last row up. Now $M'_A \cdot \mu^T = B \cdot M_A \cdot \mu$ implies $M_A \cdot \mu = 0$ (multiplying by $B^{-1}$ from the left, which can be done since $B$ is non-singular). Therefor, $R'' = R' + (\text{sec}' - \text{sec}) \cdot \mu^T \cdot \mu$ also satisfies $X = M_A \cdot R$ for all $\text{sec}' \in F$. This proves that given $X$, $R(0,0)$ appears uniformly and independently (of the other entries) distributed over $F$, and privacy follows.

Next, we prove pairwise verifiability. Let $A \in A$, and denote by $S'$ the $|A| \times e$ matrix with $S'_i = s_i$.

**Claim 3.1.** There exists a matrix $R$ (not necessarily symmetric) which satisfies $M_j \cdot s_i = M_j \cdot R \cdot M_i$ for all $i \neq j \in A$. In other words, the equation system $M_A \cdot R \cdot M_A^T = M_A \cdot S'^T$ is solvable.

Assume the claim holds. Recall we extract the effective shares $s_i^E$ from the $s_i$’s by letting $s_i^E = s_{i,1}$. By existence of a matrix $R$ as in the claim, for each $i \in A$ we have $R_1 \cdot M_i = \text{tar} \cdot R \cdot M_i = h_A \cdot M_A \cdot R \cdot M_i = 32$
In other words, \( s_{A,1} \) is consistent with a valid sharing of the “effective” scheme \( D^E \) (using randomness \( R_1 \)).

It remains to prove Claim 4.4.

**Observation 3.3.** For each row \( M_i \) in \( M_A \), it holds that if \( \alpha \cdot M_A = 0 \), for some \( \alpha \in \mathbb{F}^{|A|} \), then \( \langle \alpha, S' \rangle = 0 \) as well. In other words, all linear dependencies that would hold between rows of \( S'_i \) if the shares are distributed properly, indeed hold.

The above observation holds since \( S'_{i,j} = \langle m_j, \tilde{s}_i \rangle \) holds for all \( j \in A \) and some vector \( \tilde{s}_i \in \mathbb{F}^e \) (by definition of the scheme).

Fix a submatrix \( M'_{A'} \) of \( M_A \) which is a basis to \( M_A \)’s row space, and let \( M'_{A'} = M_{A \setminus A'} \). Likewise, denote by \( S'_{A'} \) the submatrix of \( S' \) where only the rows corresponding to \( A' \) are taken, and let \( S'_{A'} = S'_{A \setminus A'} \). We first observe that

\[
R \cdot M'_{A'} = S'_{A'}
\]

is solvable. To obtain a solution, complete \( M'_{A'} \) into a non-singular matrix, and complete \( S'_{A'} \) arbitrarily (by adding columns to each). Then multiply both sides by the inverse of \( M'_{A'} \) to obtain a solution. Multiplying both sides by \( M_A \), we conclude that

\[
M_A \cdot R \cdot M'_{A'} = M_A \cdot S'_{A'}
\]

is also satisfied by \( R \). Next, we prove that a solution to Equation 3.1 is in turn a solution to the entire system \( M_A \cdot R \cdot M'_{A'} = M_A \cdot S' \). To this end, we should prove that the entries of \( G' = M_A : S' \) (determined by \( M_A : S'_{A'} \)), are as implied by \( M_A : R \cdot M'_{A} = G \). So far we’ve proved that \( G_{A' \times A'} = G'_{A' \times A'} \) is symmetric (by the pairwise verification condition). Thus \( M_i \cdot R \cdot M'_{A'} = M_j \cdot R \cdot M'_{A'} \) for all \( i, j \in A' \). \( G_{A' \times A'} = G'_{A' \times A'} \) since \( G_{A' \times A'} = G'_{A' \times A'} \) is symmetric, and since \( M_A \) span \( M_A \). We thus conclude that \( G_{A' \times A} = G'_{A' \times A} \).

Finally, equality on the rest of the entries holds by applying Observation 4.4 to rows \( A' \).

Finally, if each \( i \in [n] \) can label more than one row, one can reduce this more general case to the case above, by uniquely relabeling rows labeled by \( i \) to \( (i, 1), \ldots, (i, n_i) \), and considering an access structure over the new set of parties, replacing each \( A \in A \) by \( A' = \{(i, j)| j \in [n_i], i \in A \} \). The construction then proceeds as described above.

\[ \Box \]

**A few concrete schemes.** Next, we include some (threshold) secret sharing schemes from the literature that we refer to in this work.

**Definition 3.4** [(\( n, t \))-Shamir secret sharing [120]] Let \( \mathbb{F} \) be a field with \(|\mathbb{F}| > n \). Let \( \{e_1, \ldots, e_n\} \in \mathbb{F} \setminus \{0\} \). The \((n, t)\) Shamir scheme over \( \mathbb{F} \) is specified by:

- \( S = \mathbb{F} \), and \( R = \mathbb{F}^t \).
- \( D(\text{sec}, r) : \) Let \( r(z) = \sum_{i=1}^t r_i \cdot z^i + s_i \), and \( s_i = r(i) \) (from now on, we will often abuse notation, and use \( i \) instead of \( e_i \)).
- \( \text{Rec}_A(s_A) \): For \( A \subseteq [n] \), \(|A| > t \) (otherwise define arbitrarily), output \( r(0) \), where \( r(z) \) is a degree-\( t \) polynomial interpolated from \( s_A \) using Lagrange interpolation.
3.2.4 Set systems

Definition 3.5. [(n, t)-CNF secret sharing [80]] Let \( F \) be a finite field. The \((n, t)\)-CNF scheme over \( F \) is defined as follows:

1. \( S = F \), and \( R = F^{(t)}-1 \).
2. \( D(\text{sec}, r) \): Order the size-\( t \) subsets of \([n]\) arbitrarily: \( A_1, \ldots, A_{\binom{n}{t}} \). Set \( g_i = r_i \) for \( i < \binom{n}{t} \), and \( g_i = s - \sum_{i=1}^{\binom{n}{t}} r_i \) for \( i = \binom{n}{t} \). Let \( s_i = (g_i)_{i \notin A_j} \).
3. \( \text{Rec}(s_A, A) \): Assume \( A \subseteq [n], |A| > t \) (otherwise, defined arbitrarily). Output \( \sum_{i=1}^{\binom{n}{t}} g_i \), where each \( g_i \) is extracted from \( s_i \) for the smallest \( i = \text{min}([n] \setminus A_j) \).

Definition 3.6. [(n, t)-Bivariate Shamir [14]] Let \( F \) be a field with \(|F| > n \). Let \( \{e_1, \ldots, e_n\} \subseteq F \setminus \{0\} \). The \((n, t)\)-Bivariate Shamir scheme over \( F \) is defined as follows:

1. \( S = F \), and \( R = F^{(t+1)-(t+1)}-1 \). It is convenient to view \( R \) as vectors indexed by \((i, j)\), where \( i, j \in \{0, \ldots, t\} \), such that not both \( i \) and \( j \) are 0.
2. \( D(\text{sec}, r) \): Let \( r(z_1, z_2) = \sum_{i=1}^{t} \sum_{j=0}^{t} r_{ij} \cdot z_i^j \cdot s + s \), and define the (degree-\( t \)) polynomials \( row_i(z) = r(i, z) \), \( col_i(z) = r(z, i) \). It outputs \( s_i \) be the sequence of evaluations of \( row_i(z) \), and \( col_i(z) \) at \( \text{Rec}(2n-1 \text{ shares overall}) \).
3. \( \text{Rec}(s_A, A) \), for \( A \subseteq [n], |A| > t \), outputs \( r(0, 0) \), where \( r(z_1, z_2) \) is a degree-\( t \) polynomial (by degree-\( t \) we mean degree-\( t \) in each variable unless stated otherwise) interpolated from \( s_A \) using Lagrange.

Clearly, all the above schemes are linear (LSSS). As explained below, all these schemes are also \( d \)-wise multiplicative for \( n > dt \). For CNF, the function \( \text{MULT} \), is defined as follows: given the shares \( \{s^i_j | l \in [d], j \in [n] \} \), let party \( i \) output \( \sum_{j_1, \ldots, j_d} \prod_{i \in [d]} g^j_{j_i} \) over all tuples \( (j_1, \ldots, j_d) \), such that party \( i \) is the party with the lowest index among those assigned \( g_{j_1}, \ldots, g_{j_d} \) by the CNF scheme (for a “generic” secret \( s \)). Clearly, if every tuple \( (j_1, \ldots, j_d) \in \binom{n}{t}^d \) is assigned to some party, we have \( \sum_{i=1}^{n} \text{MULT}(i, s^1_i, \ldots, s^d_i) = \prod_{i=1}^{d} \sum_{j=1}^{\binom{n}{t}} g^j_i \). This is indeed the case, since each \( g_j \) is given to all but \( t \) parties, so for each tuple \( g_{j_1}, \ldots, g_{j_d} \), all but the most \( d \cdot t \) parties know all these shares, thus there exist at least \( n - d \cdot t > 0 \) parties holding this tuple.

Shamir is \( d \)-multiplicative for \( n > dt \), simply by letting \( \text{MULT}(s^1_i, \ldots, s^d_i) = \alpha_i \cdot \prod_{l \in [d]} s^l_i \), where \( \alpha_i \) is the coefficient of \( s_i \) in \( h[n] \) in \((n, n)\)-Shamir. This works since \( \prod_{j \in [d]} s^j_i \) is the evaluation at \( i \) of the product of degree-\( t \) polynomials constituting the sharings of the \( s^j_i \)'s, \( p(z) \), and we have \( p(0) = \prod_{j \in [d]} s^j \). Since its degree is at most \( d \cdot t \), \( n > d \cdot t \) evaluations determine a unique degree-\( dt \) polynomial, and thus \( \prod_{j \in [d]} s^j \) is recovered correctly. Bivariate Shamir is \( d \)-multiplicative for \( n > d \cdot t \) is \( d \)-multiplicative from analogous reasons (using the fact that a \((dt+1) \times (dt+1) \) matrix of evaluations uniquely determines a degree-\( dt \) bivariate polynomial).

3.2.4 Set systems

Some of our protocols rely on set systems \( \mathcal{T} = \{T_1, \ldots, T_m\} \subseteq 2^{[n]} \), characterized by the following parameters:
- \( t \)-resilience: for every \( t \)-tuple \( H \subseteq [n] \), there are at least \( b \) sets in \( \mathcal{T} \) which \( H \) does not intersect.
- \( h \)-largeness: every set \( T_i \in \mathcal{T} \) satisfies \( |T_i| \geq h \).
- \( h' \)-pairwise largeness: every pair of sets \( T_i, T_j \in \mathcal{T} \) satisfy \( |T_i \cap T_j| \geq h' \).

We refer to a set system with parameters as above as a \((n, t, b, h, h')\)-large system of size \( m \). We abbreviate \((n, t, b, h, 0)\)-large as \((n, t, b, h)\)-large, and \((n, t, b, 0, h') = (n, t, b, h', h')\) as \((n, t, b, h')\)-pairwise large.

Typically, we will have \( h = \Theta(t^3) \). We refer to \( m = |\mathcal{T}| \) as the size of the set system. For our purposes, a primary goal will be to make the domain size, \( n \), as small as possible as a function of the resilience parameter \( t \). A secondary goal would be to keep \( m \) as small as possible. Next, we describe several constructions of set systems starting with one that guarantees a large value for \( b \).

**Theorem 3.3.** For all \( t \geq 1 \), there exists a \((n = \Theta(t^3), t, b = m/2 + 1, h = t + 1)\)-pairwise large system of size \( m = 4t + 1 \).

**Proof.** Let \( n \geq (t + 1) \cdot \binom{m}{2} = \Theta(t^3) \). For each of the \( \binom{m}{2} \) pairs of sets \( T_i, T_j \), include \( t + 1 \) distinct elements in both sets (and in no other set). It follows that their intersection is of size \( t + 1 = h \). Moreover, any \( t \)-tuple \( H \subseteq [n] \) intersects at most \( 2t < m/2 \) sets (since each element is in exactly two sets), and so we have \( t \)-resilience. □

Allowing weaker resilience, specifically \( b = 1 \) (namely, each \( t \)-tuple avoids at least one set), while keeping the intersection parameter at \( h = t + 1 \), we can achieve a better dependence of \( n \) on \( t \):

**Theorem 3.4.** For all \( t \geq 1 \), there exists a \((n = 3t + 1, t, b = 1, 2t + 1, t + 1)\)-large set system of size \( m = \binom{3t+1}{t} \).

**Proof.** To obtain such a set system, take all subsets of \([n]\) of size \( 2t + 1 \). Since \( n = 3t + 1 \), the intersection of each pair of sets is of size at least \( t + 1 \), and each \( t \)-tuple \( H \) avoids the set \([n] \setminus H\) (which is in the set system). □

The disadvantage of this construction is that the number of sets, \( m \), is exponential in \( t \). Alternatively, we can achieve \( n = o(t^3) \) with \( m = poly(t) \), by settling for \((n, t, 1, O(t))\)-large (rather than pairwise large) systems. More concretely:

**Theorem 3.5.** For all constants \( c \), and all \( t \geq 1 \), there exists a \((n = O(t^2 / \log(t)), t, b = 1, c \cdot t)\)-large system of size \( m' = poly(t) \).

**Proof.** This is a direct corollary of [2, Theorem 4.7]. □

### 3.3 Protocols in the client-server model

In this section, we present several variants of a two-round protocol which operates in a setting where the parties consist of \( m \) clients and \( n \) servers. The clients provide the inputs to the protocol (in its first round) and receive its output (in its second round) but the “computation” itself is performed by the servers alone.
Our construction provides security against any adversary that corrupts either a single client or at most \( t \) servers. We refer to this kind of security as \((1, t)\)-security\(^9\). The protocols in this setting illustrates some of the techniques we use throughout the chapter, and it can be viewed as a warmup towards our main results. The perfect protocol in Section 3.3.1 is the simplest among them, and we do not attempt to optimize its parameters, but rather illustrate many of our techniques in a clear way.

In the following two subsections, we present several constructions which are \((1, t)\)-secure with better resilience than that achieved by our perfect construction from Section 3.3.1. Each of them achieves some tradeoff between resilience and efficiency. More specifically, the complexity of the first one depends on \( n \) exponentially, and is thus applicable only to settings with a constant (in fact, up to logarithmic) number of servers. The complexity of the second one is polynomial in all parameters, but achieves worse resilience. Both constructions are made possible by weakening our requirements from the set system (allowing systems with better dependence of \( n \) on \( t \)), at the cost of somewhat complicating the protocol.

### 3.3.1 A \( (1, \Omega(n^{1/3})) \)-secure protocol

For any functionality \( f \in \text{POLY} \), we present a 2-round \((1, t)\)-secure MPC protocols (with guaranteed output delivery) for \( m \geq 2 \) clients and \( n = \Theta(t^3) \) servers. The protocol makes a black-box use of a PRG, or alternatively can provide unconditional security for \( f \in \text{NC}^1 \).

**Tools.** Our protocol relies on the following building blocks:

1. An \((n, t)\)-secret sharing scheme for which it is possible to check in \( \text{NC}^1 \) whether a set of more than \( t \) shares is consistent with some valid secret. For instance, Shamir’s scheme satisfies this requirement. Unlike the typical use of secret sharing in the context of MPC, our constructions do not rely on linearity or multiplication property of the secret sharing scheme.

2. A set system \( \mathcal{T} \subseteq 2^{[n]} \) of size \( \ell \) such that (a) \( \mathcal{T} \) is \( t \)-resilient, meaning that every \( B \subseteq [n] \) of size \( t \) avoids at least \( \ell / 2 + 1 \) sets; and (b) \( \mathcal{T} \) is \((t + 1)\)-pairwise intersecting, meaning that for all \( T_1, T_2 \subseteq \mathcal{T} \) we have \( |T_1 \cap T_2| \geq t + 1 \). See Section 3.2.4 for a construction with \( n = \Theta(t^3) \), \( \ell = \text{poly}(n) \).

3. A PSM protocol, with the best possible privacy (according to Theorem 3.1, either perfect or computational) for some functions \( f' \) depending on \( f \) (see below).

**Perfect security with certified randomness.** We start with a protocol for \( m \geq 2 \) clients and \( n = \Theta(t^3) \) servers, denoted \( \Pi_R^T \), that works in a scenario where each set of servers \( T \in \mathcal{T} \), shares a common random string \( r_T \) (obtained in a trusted setup phase). We explain how to get rid of this assumption later.

- **Round 1:** Each Client \( i \) secret-shares its input \( x_i \) among the \( n \) servers using the \( t \)-private secret sharing scheme.

- **Round 2:** For each \( T \in \mathcal{T} \) and \( i \in [m] \), the set \( T \) runs a PSM protocol with the shares \( s \) received from the clients in Round 1 as inputs, \( r_T \) as the common randomness, and Client \( i \) as the referee (i.e.,

---

\(^9\)Recall that the impossibility results of [56] imply that general 2-round protocols in this setting tolerating a coalition of a client and a server are impossible.
one message is sent from each server in $T$ to Client $i$). This PSM protocol computes the following functionality $f'_i$: 
- If all shares are consistent with some input value $x$, then $f'_i(s) = f(x)$.
- Else, if the shares of a single Client $i$ are inconsistent, let $f'_i(s) = \bot$.
- Otherwise, let $j$ be the smallest such that the shares of Client $j$ are inconsistent. Then, $f'_i(s)$ is an “accusation” of Client $j$; i.e., a pair $(j, f(x'))$, where $x'$ is obtained from $x$ by replacing $x_j$ with 0.

- **Reconstruction:** Each Client $i$ computes its output as follows: If all sets $T$ blame some Client $j$, then output the (necessarily unanimous) “backup” output $f(x')$ given by the PSM protocols. Otherwise, output the majority of the outputs reported by non-blaming sets $T$.

Lemma 3.5. $\Pi^R$ is a 2-round, $(1, t)$-secure MPC protocol for $m > 1$ clients and $n = \Theta(t^3)$ servers, assuming that the servers in each set $T \in \mathcal{T}$ have access to a common random string $r_T$ (unknown to the clients). The security can be made perfect for $f \in \text{NC}^1$, and computational for $f \in \text{POLY}$ by making a black-box use of a PRG.

**Security proof idea.** If the adversary corrupts at most $t$ servers (and no client), then privacy follows from the use of a secret sharing scheme (with threshold $t$). By the $t$-resilience of the set system, a majority of the sets $T \in \mathcal{T}$ contain no corrupted server and thus will not blame any client and will output the correct value $f(x)$.

If the adversary corrupts Client $j$, then all servers are uncorrupted. Every set $T \in \mathcal{T}$ either does not blame any client or blames Client $j$. Consider two possible cases: (a) Client $j$ makes all sets $T$ observe inconsistency: in such a case, Client $j$ receives $\bot$ from all $T$ and hence does not learn any information; moreover, all uncorrupted clients will output the same backup output $f(x')$. (b) Client $j$ makes some subsets $T$ observe consistent shares: since the intersection of every two subsets in $\mathcal{T}$ is of size at least $t + 1$ then, using the $(t + 1)$ reconstruction threshold of the secret sharing scheme, every two non-blaming sets must agree on the same input $x$. This means that Client $j$ only learns $f(x)$. Moreover, all other (uncorrupted) clients will receive the actual output $f(x)$ from at least one non-blaming set $T$ and, as discussed above, all outputs from non-blaming sets must agree.

Observe that the fact that a set $T$ uses the same random string $r_T$ in all $m$ PSM instances it participates in does not compromise privacy. This is because in each of them the output goes to a different client and only a single client may be corrupted.\footnote{Alternatively, $r_T$ can be made sufficiently long so that the set $T$ can use a distinct portion of $r_T$ in each invocation of a PSM sub-protocol.}

A rigorous proof of Lemma 3.5 appears in Section 3.3.1.1.

**A note on the PSM used.** The protocol’s specification does not spell out the privacy level and assumptions for the PSM used, but it is implicitly assumes the best achievable parameters for $f$, while keeping the PSM protocol efficient. If $f \in \text{NC}^1$, then the $f'_i$’s evaluated by the various PSM executions are as well (computing $f'_i$ involves evaluation of $f$, and Lagrange interpolation on shares of length $O(|x|)$, which is also in $\text{NC}^1$). Thus, by Theorem 3.1, a perfectly private PSM can be used. For general $f \in \text{POLY}$, the $f'_i$’s are in $\text{POLY}$ as well, so a computationally private PSM, using a PRG in a black-box manner can be used (again, by Theorem 3.1).
Removing the certified randomness assumption. If we have at least 4 clients, we can let each Client $i$ generate its own candidate PSM randomness $r^i_T$ and send it to all servers in $T$. Each PSM protocol, corresponding to some set $T$, is executed using each of these strings, where in the $i$-th invocation (using randomness $r^i_T$) Client $i$ receives no message (otherwise, the privacy of the protocol could be compromised). The other clients receive the original messages as prescribed by the PSM protocol. Upon reconstruction, Client $i$ lets the PSM output for a set $T$ be the majority over the $m - 1$ PSM outputs it sees. We observe that this approach preserves perfect security. If statistical (or computational) security suffices, we can rely on 2 or 3 clients as well. Here, the high level idea is to use a transformation as described for the case $m \geq 4$, and let Client $i$ authenticate the consistency of the randomness $r^j_T$ used in the $j$'th PSM protocol executed by set $T$, using a random string it sent in Round 1. Upon reconstruction, each client only considers the PSM executions which passed the authentication. Combining the above discussion with Theorem 3.1, we obtain the following theorem.

**Theorem 3.6.** There exists a statistically $(1, t)$-secure 2-round general MPC protocol in the client-server setting for $m \geq 2$ clients and $n = \Theta(t^3)$ servers. For $f \in \text{NC}^1$, the protocol is perfectly secure if $m \geq 4$, and statistically secure otherwise. The protocol is computationally secure for $f \in \text{POLY}$. In all cases, the simulation is straight-line.

A formal specification of the transformation along with a proof of the theorem appears in Section 3.3.1.2.

### 3.3.1.1 Proof of Lemma 3.5

We describe a straight-line simulator $S^R$ to simulate the view of a malicious adversary $\mathcal{A}$. We prove security for the perfect case (and $f \in \text{NC}^1$). We assume without loss of generality that the computationally-unbounded adversary is deterministic (otherwise, its randomness can be viewed as a random input given by the environment to both the adversary and the simulator). The security proof for the computational case (and $f \in \text{POLY}$) is practically identical, referring to a computationally private, rather than a perfectly private PSM. In particular, in the latter case, we obtain computational security if a client is corrupted, and statistical security in case $\leq t$ servers are corrupted. In the following proofs, we slightly abuse notation, and sometimes refer to $\mathcal{A}$ as the set of indices of corrupted parties.

**Corrupted servers.** Assume $|\mathcal{A}| = t' \leq t$. The simulator sends nothing to the functionality. The adversary’s view is simulated as a sequence of (partial) sharings $v_1, \ldots, v_m$, where each $v_i$ is independently sampled from the distribution of partial sharings to $\mathcal{A}$ of, e.g., 0.

The simulation is perfect by the privacy of the secret sharing scheme. By the simulators’ construction, the output of uncorrupted parties in the ideal world is $f(x)$ with probability 1. This is also the case in the real world since a (strict) majority of sets does not contain malicious parties (by largeness of the set system), so the PSM output of all these sets is $f(x)$ (in particular, all these sets are necessarily non-blaming).

**A corrupted client.** Assume the adversary $\mathcal{A}$ corrupts Client $j$. The simulator $S^R$ proceeds according to the steps of the protocol, as follows:

- **Round 1:** Feed the adversary with $x_j$, and observe its outgoing Round 1 messages (there are no incoming messages in Round 1).
- **Calling the functionality**: Extract from the adversary’s messages an effective input \( x_j^* \) as follows:
  1. If no one of the sets \( T \) obtains a consistent sharing of any value, then \( x_j^* = 0 \).
  2. Otherwise, let \( T \) be the first set (according to some pre-defined ordering) that obtains a consistent sharing of some value \( v^* \), and set \( x_j^* = v^* \). Simulator \( S^R \) calls the functionality with input \( x_j^* \) and obtains an output \( out \).

- **Round 2**. Let \( S \) denote the (perfect) simulator guaranteed by PSM privacy. Simulate the output of each set \( T \) that received a consistent sharing by invoking \( S(1^{[k_j]}, out) \), and the output of each other set \( T \) by invoking \( S(1^{[k_j]}, \bot) \) (executed with fresh randomness for every such set).

By definition of \((1, t)\)-security, if a client is corrupted, all servers are uncorrupted. We analyze cases (1), (2) described above. In both cases, the adversary’s view is simulated correctly by privacy of the PSM protocol (note that although the randomness \( r_T \) shared by a set \( T \) is reused for all of its PSM executions involving different clients as the referee, there is no problem, since the adversary sees only one such execution, as it corrupts only one client). Next, we prove robustness, by showing that for any view of the adversary, the outputs of uncorrupted players in the real and ideal worlds are identically distributed.

- **Case (1)**. The uncorrupted clients in the real execution output \( f(x_j^* = 0, x_{[n]\setminus\{j\}} = x_{[n]\setminus\{j\}}) \) (by construction and correctness of the PSM). In this case \( S^R \) submits \( x_j^* = 0 \), so this is the uncorrupted clients’ output of the ideal execution as well.

- **Case (2)**. The crucial observation is that, since the sets \( T \) belong to a \( t + 1 \) pairwise-large system (in the real execution), there exists a value \( v^* \), so that for all sets holding a consistent (partial) sharing of \( x_j \), the sharing is consistent with \( v^* \) (since \( t + 1 \) shares determine the shared value). Therefore, uncorrupted clients in the real execution will obtain \( f(x_j^* = v^*, x_{[n]\setminus\{j\}} = x_{[n]\setminus\{j\}}) \) from \( T \), and get no contradicting values from other non-blaming sets. By the same reasoning, the simulator submits \( v^* \) as its input and uncorrupted clients output \( out = f(x_j^* = v^*, x_{[n]\setminus\{j\}} = x_{[n]\setminus\{j\}}) \).

We stress that the PSM-related arguments in the above proof crucially rely on the fact that servers in the real execution share a random string \( r_T \) (obtained in a trusted setup phase).

### 3.3.1.2 Waiving the certified randomness assumption

We start with formally stating the transformation from a statistically (respectively, computationally) \((1, t)\) secure protocol \( \Pi^R \) in the setting with certified randomness into a statistically (respectively, computationally) secure protocol in the plain setting. Namely, we prove the following lemma.

**Lemma 3.6.** Let \( \Pi^R \) be a \((1, t)\)-secure \( m \)-client \((m > 1), n\)-server protocol as in Section 3.3 (using certified PSM randomness) for evaluating \( f \). Then there exists a \((1, t)\)-secure \( m \)-client, \( n \)-server protocol \( \Pi \) for evaluating \( f \) in the plain model, where \( \Pi \) is statistically secure for \( f \in \text{NC}^1 \) and computationally secure for \( f \in \text{POLY} \).
A proof sketch. Below, we only describe the transformation for $m = 2, 3$ clients (the transformation for the simpler case $m \geq 4$ is sketched in Section 3.3). Additionally, we only present a proof for the statistical case (that is, transforming the perfectly secure $\Pi^R$). The proof for the computational case (transforming a computationally secure $\Pi^R$) is very similar, and is omitted. The high level idea is to modify the construction for $m > 3$, so that the PSM output by a set $T$ of servers to Client $i$ comes along with an “authentication” string, allowing to check that the randomness used for the PSM (sent by a different client) is consistent. This is required here because there are not enough clients to guarantee that a majority of PSM executions by a set run with consistent randomness.

Construction details. We start with describing some tools we will use.

Definition 3.7. Let $X$ be a random variable distributed over a finite domain $D$. The min-entropy of $X$, $H_\infty$ is defined as $H_\infty(x) = \min_{d \in D} \log(\Pr[X = d]^{-1})$.

Definition 3.8. [(explicit) strong extractor [119]] For functions $k(n), \epsilon(n), r(n), m(n)$ a family $\text{EXT} = \{\text{EXT}_n\}$ of polynomially computable functions $\text{EXT}_n : \{0, 1\}^n \times \{0, 1\}^r \rightarrow \{0, 1\}^m$ is an explicit $(k, \epsilon)$-extractor if for all $n$, and any distribution $X$ on $\{0, 1\}^n$ with $H_\infty(X) \geq k$, $(Y, \text{EXT}_n(X, Y))$ (Y is distributed according to $U_{\{0, 1\}^r}$) is $\epsilon$-close to $U_{\{0, 1\}^m}$.

We will use strong extractor with $k = \Omega(n), \epsilon n = e^{-\Theta(k)/\log n}$ extractors $m = \Theta(k)/\log n$ (for more details on the construction, see [119]).

Construction 3.4. Given a finite field $F$, and integer $l > 0$, let $H(F, l)$ denote a family of hash functions specified by vectors in $F^l$; for a vector $v \in F^l$, the function associated with $v$ is $H(v, x) = \sum_{j=1}^l v_j \cdot x_j$.

Construction 3.5. We proceed as in $\Pi^R$ with the following modifications.

- Round 1: For a set $T$, we interpret the PSM randomness $r_T$ as a length-$l = |T| + 1$ vector over a sufficiently large field, of size $q = 2^{O(k)}$. Client $i$ sends each member in $T$ a random independent vector $r_T^i$, a seed $s_T^i$ for a strong extractor $\text{EXT}$ as above, with $n = \log q \cdot l, k(n) = \log q$, and a hash function picked from $H(F, l)$ independently at random.

- Round 2: For each $i \in [m]$, each set $T$ runs an independent PSM execution against each client $j \neq i$ (computing the same $f'_T$ as before), using randomness $\text{EXT}(r_T^i, s_T^i)$. Additionally, in such an execution server $l$ sends its PSM output $A_T$ to Client $j$ along with $s_T^i, h_T^i(r_T^i)$. (Overall, each client receives $m - 1$ “authenticated” PSM outputs).

- Reconstruction: For each set $T$, Client $i$ first finds some $j$ for which all servers in $T$ send equal values of $h_T^i(r_T^j)$’s and of $s_T^i$’s (in the sequel, we refer to this situation as “having consistent randomness”). If none exists, “disqualify” $T$. Otherwise, interpret $T$’s $j$-th PSM output as $T$’s output, $out_T$. If all the sets were disqualified, output $f(x')$ where $x'_i = x_i, x'_{[n] \setminus i} = 0$.$^{11}$

Otherwise, let $\text{Good}$ denote the collection of “surviving” sets. Otherwise, proceed like $\Pi^R$ as if $\bigcup_{T \in \text{Good}} T$ was the set of servers, and $\text{Good}$ is the set system employed (using the $out_T$’s as the sets’ outputs).

$^{11}$In fact, the latter can only happen for $m = 2$. 40
To prove security, we show how to modify $S^R$ in each case.

**Corrupted servers case.** Assume $t' \leq t$ servers are corrupted. The simulator in this case is identical to $S^R$, except that to simulate the adversary’s view, for each $T$ affected by the adversary, for each $i \in [m]$ we append $h^i_T, r^i_T, s^i_T$ selected independently at random from the distribution corresponding to Client $i$’s incoming Round 1 message. Clearly, we perfectly simulate the adversary’s view. Now, we prove that the clients output $f(x)$ with probability 1. By resilience of the set system (and the fact that all clients are uncorrupted), a strict majority of sets “survive disqualification” (by each of the clients) and return $f(x)$. Therefore, $f(x)$ is voted by a majority of “surviving”, non-blaming sets, and is therefore picked by the majority “vote”, and is the output of each client.

**Corrupted client case.** For Client $j$ the simulator $S$ is as follows.

- **Round 1:** $S$ feeds $A$ with $x_j$, and obtains its Round 1 messages.
- **Calling the functionality:** If $m = 3$ act like $S^R$, applied to the shares sent to the various sets (disregarding the randomness part of the messages). If $m = 2$, let Good denote the collection of “surviving” sets. If Good is empty, submit a 0 as input. Otherwise, act like $S^R$, as if the set of servers is $\bigcup_{T \in \text{Good}} T$, and $T = \text{Good}$. That is, disregarding $A$’s messages to the other sets, feed $S^R$ with the shares sent by $A$ to each of the sets in Good. Submit the functionality the same messages as $S^R$.

- **Round 2:** Let $\text{out}$ denote the functionality’s output, and let $S$ denote the simulator guaranteed by PSM privacy. For each set $T$, to simulate its Round 2 messages, pick a random independent value $s^i_T$ for each $i \in [m] \setminus \{j\}$, and output $(S(1^{[x_j]}, \text{out}, s^i_T)_{i \in [m] \setminus \{j\}}$ if $A$ sent $T$ consistent shares and $(S(1^{[x_j]}, \bot, s^i_T)_{i \in [m] \setminus \{j\}}$ otherwise.\footnote{We stress that each execution of $S$ uses fresh randomness.}

We will need the following technical observations.

**Observation 3.4.** For $t \geq 1, v_1 \neq v_2 \in \mathbb{F}_q^t$, we have $\Pr_{v \in \mathbb{F}_q^t}[H(v, v_1) \neq H(v, v_2)] = q^{-1}$.

**Observation 3.5.** For $t \geq 1, v_1, v_2, \ldots, v_t \in \mathbb{F}_q^{t+1}$, if the equation system $Vx = b$ (where $v_i$ is the $i$-th row of $V$) is solvable, then its set of solutions is an affine subspace of $\mathbb{F}_q^{t+1}$ of rank $\geq 1$ (and thus of size $\geq q$).

The following is a direct corollary from Observation 3.4:

**Corollary 3.7.** In the real execution, if Client $j$ sends inconsistent randomness to set $T$, then the other clients will identify the $j$’th PSM execution by $T$ as having inconsistent randomness, with overwhelming probability.

We first argue that the adversary’s view is statistically simulated. This holds since the uncorrupted clients, send the same random and independent (of other values sent) $r^i_T, s^i_T$ to all parties in $T$, for each set $T$. For each $j \neq i \in [m]$, let $h_{1,T}, \ldots, h_{|T|,T}$ denote the hash functions sent by Client $i$ to set $T$. From
T’s outputs for the $j$-th execution the adversary learns $h_{1,T}(r_{T}^{j}), \ldots, h_{|T|,T}(r_{T}^{j})$, along with $s_{T}^{j}$. It follows from Observation 3.5 that $r_{T}^{j}$ has min-entropy $\theta(k)$. Taking EXT to be a strong extractor as guaranteed above with suitable parameters ($m(n) = r$, where $r$ is the length of randomness used by the PSM), it follows that $\text{EXT}(r_{T}^{j}, s_{T}^{j})$ is statistically close to uniform, given all that information, with overwhelming probability (over the choice of $s_{T}^{j}$). Combining PSM privacy with this fact, we conclude the simulated view is statistically indistinguishable from the real view.

The robustness of the protocol is argued next. More concretely, we show that for any specific view of the adversary, the output of the uncorrupted parties in the real and ideal worlds are statistically indistinguishable, and the claim follows. There are two cases.

- Assume $m = 3$. In the ideal world, the adversary’s submitted input (and thus uncorrupted client’s output) is distributed exactly as that of $S^{R}$, where the adversary’s shares are as $A$’s. In the real world, the uncorrupted clients’ output is statistically close to their output in $\Pi^{R}$, where the adversary sends the same shares as $A$. The last claim follows by the following two observations, combined with the construction details of $\Pi$. (1) Each client receives a PSM output based on consistent randomness (sent by another uncorrupted client) from each $T$. (2) By Corollary 3.7, an uncorrupted client interprets a PSM output based on inconsistent randomness (sent by Client $j$) as non-blaming and different from $f(x)$ is negligible. Finally, since $S^{R}$ is a “proper” simulator for $\Pi^{R}$, the real and ideal distributions here are also statistically close.

- Assume $m = 2$. If all sets $T$ receive inconsistent randomness according to $A$, $S$ submits 0 to the functionality. In the ideal world, the other clients outputs $f(x')$ (where $x'$ is as defined in the protocol above) with probability 1. In the real world, by Corollary 3.7, an uncorrupted client disqualifies each set with overwhelming probability, and by the union bound, it disqualifies all sets with overwhelming probability (for a sufficiently large field size $|F|$), and outputs $f(x')$. Otherwise, in the ideal world, the simulator sends the same messages as $S^{R}$ restricted to the non-empty set Good, when fed the shares sent by $A$. In the real world, each client outputs what $\Pi^{R}$ would restricted to Good, for an adversary sending to Good the same shares as $A$ (again, using Corollary 3.7 and the union bound). To complete the proof, it remains to show that $S^{R}$ properly simulates $\Pi^{R}$ for any subset Good $\subseteq \mathcal{T}$. The main observation here is the fact that the validity proof of $S^{R}$ relies only on the $t + 1$ pairwise-largeness of $\mathcal{T}$, which is preserved by subsets, and does not rely on the resilience property.

3.3.2 An $n = 3t + 1$ server $(1, t)$-secure construction (inefficient)

In this section, we show how to improve $t$ in the $(1, t)$-security from ($t = \Theta(n^{1/3})$) to $t = \lfloor(n - 1)/3 \rfloor$.

More precisely, we prove the following.

**Theorem 3.8.** There exists a $(1, t)$-secure general 2-round MPC protocol in the client-server setting for $n > 3t$ servers and $m > 1$ clients. The protocol provides statistical security for $f \in \text{NC}^1$ and computational security for $f \in \text{POLY}$ by making a black-box use of a pseudorandom generator. The time complexity of
the protocol depends exponentially on \( n \).\(^{13}\) (So, unfortunately, the resulting construction is only efficient for small \( n \).)

Intuitively, the gain in resilience is due to reducing the resilience requirement from \( |\mathcal{T}|/2 + 1 \) to \( 1 \), which allows to devise a set system with better dependence of \( n \) on \( t \). To compensate for this relaxation in resilience, we use a stronger version of PSM.

To prove the theorem, we present a \((1, t)\)-statistically secure protocol \( \Pi^R_{hyp-res} \) for the setting with certified randomness \( m > 1, n > 3t \). It can be transformed into a protocol in the plain setting by an easy adaptation of the transformation in Section 3.3.1.2 from \( \Pi^R \) to \( \Pi \). Additionally, we consider only the statistical case (as before, the claim about \( f \in POLY \), and computational efficiency is proved by plugging in a computationally private PSM).

**Construction 3.6.** The construction proceeds as \( \Pi^R \), except for the following differences.

- Replace the set system by an \((n, t, 1, 2t + 1, t + 1)\)-large set system, and partition the parties accordingly. See Section 3.2.4 for a construction of such a system.
- Use a statistically-robust, perfectly private PSM (previously, we did not require robustness).
- Reconstruction: Each Client \( i \) considers only sets that output a non-\( \perp \) value (by 1-resilience, there exists at least one such set). If all these sets blame some client, it outputs the “backup” value \( f(x') \) included in the PSM output of the first such set. Otherwise, output the PSM output \( f(x) \) of some set whose output is non-blaming (and differs from \( \perp \)).

To prove security, we show how to simulate the behavior of a computationally unbounded (and, without loss of generality, deterministic) adversary \( A \) in each case. We refer to this simulator as \( S^R_{hyp-res} \).

**Corrupted servers.** The simulator is identical to \( S^R \) (note that the description of \( S^R \) is independent of the concrete set system \( \mathcal{T} \) used). The adversary’s view is perfectly simulated (by the same reasoning as for \( S^R \) in \( \Pi^R \)). Next, we prove that for any view of \( A \) in both worlds, the distributions of the clients’ outputs are statistically indistinguishable. By construction of \( S^R_{hyp-res} \), in the ideal world, all clients output \( f(x) \) with probability 1. In the real world, each client outputs \( f(x) \) with overwhelming probability. The crucial observation is that the adversary cannot modify a sets’ PSM output to Client \( i \) to a different (from \( f(x) \)) non-blaming value, except for possibly with negligible probability. It immediately follows that (with overwhelming probability), for all non-blaming sets each client sees output \( f(x) \), and by 1-resilience of the set system, there is at least one such set.

It remains to prove the observation. For a set \( T \), which contains corrupted parties, denote by \( B \) the set of corrupted parties in \( T \). We invoke the simulator guaranteed by statistical robustness on the randomness \( R \) and outgoing messages to obtain a distribution \( x^*_B \) on the malicious parties “effective input”, so that the output of uncorrupted parties in the real protocol is statistically close to \( \text{Rec}(x^*_B, x_{T \setminus B}) \). Recall that if \( x^*_i = \perp \) for some \( i \in B \), we have \( \text{Rec}(x^*_B, x_{T \setminus B}) = \perp \) and \( \text{Rec}(x^*_B, x_{T \setminus B}) = f(x^*_B, x_{T \setminus B}) \) otherwise.

\(^{13}\)Unless mentioned explicitly, as done here, the complexity of our constructions depends polynomially on all parameters, including the number of parties.
(by correctness of the PSM). If \( x_B^* \) does not contain \( \perp \)'s, the PSM output will either contain a blaming of a client, or remain unchanged. This is so since by \((2t + 1)\)-largeness, the underlying set contains \( t + 1 \) uncorrupted parties, determining the values of all inputs of the clients, assuming \( f_i' \) on these (purported) shares is non-blaming (by correctness of the secret sharing scheme). In particular, modifying at most \( t \) of the shares (of some value \( x_h \)) either results in a valid sharing of the same value, or in an invalid sharing. That is, the output that each client sees from a set \( T \) containing corrupted servers is distributed over \( \{ f(x), \perp, \text{blaming of some client} \} \) (except, possibly, with a negligible probability).

**A corrupted client.** The analysis is very similar to the perfect case (in particular, we obtain perfect simulation for this kind of adversary).

\[ \text{Remark 3.1.} \text{ As previously mentioned, the above construction has exponential in } n \text{ work complexity (for the optimal resilience we can achieve). The concrete set system we use is the set of all } 2t + 1 \text{-sized subsets of } [3t + 1], \text{ which results in } \binom{3t+1}{t} = n^{θ(n)} \text{ sets. Furthermore, it is easy to see that every set system of size } n = Θ(t) \text{ satisfying our requirements is of size exponential in } n. \text{ This implies that we ca not obtain an efficient protocol (in all parameters, including } n) \text{ with linear resilience by merely improving the set system’s parameters, and different techniques should be sought.} \]

### 3.3.3 An \( n = O(t^2 / \log(t)) \) server \((1, t)\)-secure construction (efficient)

To be able to reduce \( n \) (compared to the construction in Section 3.3.1), while keeping the number of sets polynomial in \( n \), we further relax the requirements on the set system, and eliminate the pairwise-largeness requirement. More concretely, using a set system which is \( Θ(t) \)-large, but not necessarily pairwise large. The perfect construction, and the “inefficient” statistical construction above relied on pairwise largeness to guarantee security against a malicious client. Removing it may cause a situation where the shares a malicious party sends to two different sets are consistent with different polynomials, and allow it to learn the evaluation of \( f \) at more than one point. To avoid this, we adopt ideas from Section 3.5, and use a MCDS-style procedure which verifies that each sharing to the servers is *globally* consistent. More concretely, we prove.

**Theorem 3.9.** There exists a \((1, t)\)-secure general 2-round MPC protocol in the client-server setting for \( n > \Omega(t^2 / \log(t)) \) servers and \( m > 1 \) clients. Statistical security can be achieved for \( f \in \text{NC}^1 \), and computational security can be achieved for all \( f \in \text{POLY} \), making black-box use of a PRG.

**Proof sketch.** Again, we only sketch the statistical construction for \( f \in \text{NC}^1 \), assuming the servers share a random string (used both for PSM, and other primitives we employ that require shared randomness). It is possible to get rid of the latter assumption by an easy adaptation of techniques from Section 3.3.1.2.

**A building block - MCDS.** On a high level, we need a CDS procedure that discloses a secret under the condition that a pair of bivariate polynomials \( R(x, y), C(x, y) \), of degree \((n - 1, t)\) (that is, degree \( t \) in \( y \), and
degree \( n - 1 \) in \( x \), and \((t, n - 1)\) respectively are “sufficiently close” to a common degree-\((t, t)\) polynomial \( Q(x, y) \). More concretely, the MCDS is defined as follows.

**Definition 3.9.** A MCDS protocol has the flow of a PSM protocol for a specific, partially specified, randomized functionality, and somewhat different security requirements (in particular, the functionality is “partial”, in the sense that we do not have specific requirements on the referee’s output for some of the inputs). Namely, it is a non-interactive protocol involving \( n \) servers, who share a common random string \( r \) and an external referee, who has no access to \( r \). We have \( n \) message algorithms \( A_1(x_1, r), \ldots, A_n(x_n, r) \), and a reconstruction algorithm \( \text{Rec} \). Server \( i \) sends the referee the value \( A_i(x_i, r) \). The referee outputs \( \text{Rec}(A_1, \ldots, A_n) \).

The inputs to servers are defined as follows. Let \( E \subseteq \mathbb{F} \setminus \{0\} \) be a set of size \( n \) (for simplicity, we abuse notation, and identify between \( E \) and \([n]\)). Let \( R(x, y) \) be a degree-\((n - 1, t)\) polynomial, and \( C(x, y) \) a degree-\((t, n - 1)\) polynomial. Server \( i \) holds (the degree-\( t \)) polynomials \( r_i(z) = R(i, z) \) and \( c_i(z) = C(z, i) \). Also, there is a designated portion \( t' \) referred to as the “secret”. The protocol has the following security properties.

- **Correctness:** Assume \( R(x, y) = C(x, y) \). Then the referee outputs \( s \) for any adversary corrupting \( t' \leq t \) of the servers.

- **Secrecy:** Assume the servers are uncorrupted, and that there exists a submatrix \( M = X \times [n] \subseteq [n] \times [n] \) with \(|X| \geq 2t + 1\) such that for all \( x \in X \), we have \(|\{y \in [n] | R(x, y) \neq C(x, y)\}| \geq 2t + 1\). Then the distribution \((A_1, \ldots, A_n, R(x, y), C(x, y))\), where \( A_1, \ldots, A_n \) are the messages received by the referee for servers’ inputs \( R, C \), is independent of \( s \).

We present an implementation of the MCDS, and prove it has the required properties.

- The servers jointly pick a random degree-\((n - 2t - 1)\) polynomial \( s(z) \) with \( s(0) = s \) (using their shared randomness), and Server \( i \) picks a random independent degree-\((n - 2t - 1)\) polynomial \( p_i(z) \) with \( p_i(0) = s(i) \).

- \( A_i(\cdot) \) is defined as follows:
  - For each \( j \in [n] \), set \( p'_i(j) = R(i, j) \cdot r_{i,j} + z_{i,j} + p_i(j) \).
  - Set \( A_i = (p'_i(z), (m_{j,i} = -C(j, i) \cdot r_{j,i} - z_{j,i})_{j \in [n]}) \) where \( p'_i(z) \) is specified as a sequence of evaluations at \([n] \), and the \( r_{i,j}, z_{i,j} \)'s are designated portions of the common randomness string \( r \).\(^{15}\)

- **Rec:**
  - For each \( j \in [n] \), let \( a_i(j) = p'_i(j) + m_{i,j} \). If \( a_i(j) \) is at distance at most \( t \) from a degree-\((n - 2t - 1)\) polynomial \( a'_i(z) \), set \( a_i = a'_i(0) \) (finding \( a'_i(z) \) can be done efficiently, using Berlekamp-Welch \([18]\)). Otherwise, set \( a_i = 0 \).
  - If \( g'(z) = o_2 \) is \( t \)-close to a degree-\((n - 2t - 1)\) polynomial \( g(z) \), output \( \text{out} = g(0) \). Otherwise, output \( 0 \). (\( 0 \) is arbitrary, the concrete output in this case is not important.)

\(^{14}\)Jumping ahead, MCDS is also defined and used in Section 3.5.1, but the MCDS notion defined in this section differs from it in some technical aspects, such as requiring shared randomness.

\(^{15}\)The same idea was employed in MCDS from Section 3.5.1 to reveal a secret conditioned on the equality of two values two parties, \( A \) and \( B \), hold.
Correctness proof. For each \((i, j)\), let \((v_{i,j}^1, v_{i,j}^2) = (p'_i(j), m_{j,i}) = (r_{i,j} \cdot R(i, j) + z_{i,j} + p_i(j), -r_{i,j} \cdot C(i, j) - z_{i,j})\) the values received by the referee (from Servers \(i, j\)). We observe that:

1. If \(R(i, j) = C(i, j)\), and Servers \(i, j\) are uncorrupted, then \((v_{i,j}^1, v_{i,j}^2)\) appear to the adversary as a random solution to the equation \(v_{i,j}^1 + v_{i,j}^2 = p_i(j)\), even given \(R(x, y), C(x, y)\) (independent of all other received values).

2. If \(R(i, j) \neq C(i, j)\), and Servers \(i, j\) are uncorrupted, then \((v_{i,j}^1, v_{i,j}^2)\) appear to the referee as random values, even given \(R(x, y), C(x, y)\) (otherwise independent of all other received values).

Denote the set of uncorrupted servers by \(G\). As for correctness, by the above, for all \(i, j \in G\), we have \(v_{i,j}^1 + v_{i,j}^2 = p_i(j)\). Therefore, \(a_i(z)\) recovered by the referee is at most \(t\)-far from \(p_i(z)\), and will be properly corrected to \(a'_i(z) = p_i(z)\) (which is strictly the closest polynomial to it, as the code distance of \((n, n - 2t - 1)\) Reed-Solomon is \(2t + 1\)). Therefore, \(g'(z)\) is within \(2t\) of \(s(z)\) (corresponding to rows \(r_i(z)\) held by corrupted servers), and \(s = s(0)\) is properly recovered (by arguments similar to the above).

As to secrecy, we claim that for the following “simulator” \(S\), the random variables

\[
(s, A_1(R, C), \ldots, A_n(R, C)) \text{ and } (s, S(R, C))
\]

are identically distributed for all \(R, C\) satisfying the secrecy precondition (secrecy is immediate):

1. For each \(i \in [n]\), pick a degree-\((n - 2t - 1)\) polynomial \(\tilde{p}_i(z)\) independently at random.
2. For each \((i, j) \in [n] \times [n]\), if \((i, j) \notin M\), simulate \(v_{i,j}^1, v_{i,j}^2\) to be a random solution to the equation \(x_1 + x_2 = p'_i(j)\). Otherwise, simulate \(v_{i,j}^1, v_{i,j}^2\) be a random pair of values.

We prove that for any \(s\) and \(R, C\) satisfying the secrecy precondition, it holds that \(A(R, C), S(R, C)\) are identically distributed. Fix some secret \(s\). Consider the real execution, and fix \(s\). By the above observation, for \(i \in X\), \((v_{i,j}^1, v_{i,j}^2)\) appears as a random pair of values, independent of \(p\) for at least \(2t + 1\) values of \(j\), and as a random solution to \(x_1 + x_2 = p_i(j)\) for \(j \in [n] \setminus \{i\}\). Since \(|[n] \setminus X| \leq n - 2t - 1\), the evaluations of a degree-\((n - 2t - 1)\) polynomial \(p_i(z)\) appear on \([n] \setminus X\) as random independent values, the joint simulated view for such rows \(i\) is distributed as in the real execution. In particular, it is independent of \((p_i(0))_{i \in X}\) (in other words, no information is revealed about \(p_i(0)\) for \(i \in X\)). Similarly, \(s(z)\) on \([n] \setminus X\) is a random vector, independent of \(s(0) = s\). We conclude that each \(\tilde{p}_i(z)\) is distributed exactly as \(p_i(z)\) for all \(i \in [n]\).

Combined with the above technical observation, it follows that \(S\) is a perfect simulator for the MCDS in this case.

The construction. We are now ready to describe our construction. On a high level, it builds upon the construction from Section 3.3.2 (in particular, it also uses statistically robust PSM). It modifies the PSM to send each party the same output as before, but mask “non-blaming” replies to Client \(i\) by a random value, which is in turn, revealed under the condition that Client \(i\)’s global sharing is not “harmfully inconsistent” (in the sense that it would allow the adversary to learn extra information in the original protocol).

Construction 3.7.
• We partition the servers according to a \( n = \Theta(t^2 / \log(t)), t, 1, 8t \)-large set system (see Section 3.2.4 for a construction of such a system). The servers share a random string \( r \) comprised of:

  – For each \( i \in [m], T \in \mathcal{T} \), all servers share a common random independent value \( r_{i,T} \), formatted as randomness for execution \((i, T)\) of MCDS.

  – Every set \( T \) shares a common random string \( r_T \), used for the PSM executions.

• **Round 1:** Each Client \( i \) shares its input \( x_i \) among the \( n \) servers using the \((n, t)\) bivariate Shamir secret sharing scheme (see Section 3.2.3).  

• **Round 2:**

  – Each set \( T \in \mathcal{T} \) of servers runs a (robust) PSM protocol with their shares, and \((s_{i,T})_{i \in [m]}\)'s as inputs, where \( s_{i,T} \) is the “secret” part in each \( r_{i,T} \). The PSM protocol reveals the following outputs to the clients:

    If all shares are consistent with some input value \( x \), then the output to Client \( i \) is \( f(x) + s_{i,T} \).

    Otherwise, assume that the shares of Client \( i \) are inconsistent (and \( i \) is the lowest such index). Then, the output to all clients, except for Client \( i \), that receives \( \bot \), is a “blaming” pair \((i, f(x'))\), where \( x'_i = 0, x'_{[n]\setminus \{i\}} = x_{[n]\setminus \{i\}} \).

  – For each Client \( i \), and set \( T \) all servers run an instance of the MCDS above using the sharing of Client \( i \)'s value (interpreted as “row” and “column” polynomials \( R(x, y), C(x, y) \)), and \( r_{i,T} \) as the randomness, and send the outputs to Client \( i \). We refer to this MCDS execution as “execution \((i, T)\)”.

• **Reconstruction:** Client \( i \) computes its output as follows:

  – Consider only sets that output a non-\( \bot \) value (by 1-resilience, there exists at least one such set).

  – If all sets blame some Client \( j \), then output the (necessarily unanimous) backup output \( f(x') \) given by the PSM protocols.

  – Otherwise, let \( T \) be the first (according to some fixed ordering) non-blaming set \( T \), and let \( masked \) denote its PSM output. Recover \( s_{i,T} \) from the corresponding MCDS procedure, and output \( masked - s_{i,T} \).

**Proof sketch.** We describe a simulator \( \mathcal{S}_{\text{eff}}^R \) for a deterministic, unbounded \((1, t < n/3)\)-adversary \( \mathcal{A} \).

**Corrupted servers.** Assume \( t' < t \) servers are corrupted by an (unbounded, deterministic) adversary \( \mathcal{A} \). The simulator behaves exactly as \( \mathcal{S}^R \). That is, it sends nothing to the functionality. The adversary’s view is simulated as a sequence of (partial) sharings \( v_1, \ldots, v_m \), where each \( v_i \) is independently sampled from the
distribution of partial sharings to \( A \) of, e.g., 0. This perfectly simulates the adversary’s view (by privacy of the underlying secret sharing scheme).

Next, we argue the robustness of \( S_{\text{eff}}^R \). In the ideal model, all clients output \( f(x) \) with probability 1. By resilience of the set system, there exists at least one set that sends a “proper” output \( f(x) + s_{i,T} \) to each client \( P_i \). By correctness of the MCDS procedure, we conclude that \( s_{i,T} \) is correctly recovered by each client \( P_i \). The precondition of the MCDS correctness guarantee holds since all \( R_i(x, y), C_i(x, y) \)’s sent by clients are consistent with a degree-\((t, t)\) bivariate polynomial, and at most \( t'<t \) might not follow the MCDS protocol. No conflicting non-blaming (and non-\( \bot \)) outputs are received from other sets with overwhelming probability (by robustness of the PSM), so \( P_i \) outputs \( f(x) + s_{i,T} - s_{i,T} = f(x) \) with overwhelming probability (see security proof of \( \Pi_{\text{hyp-res}}^R \) for a more formal form of the last argument).

**A corrupted client.** Assume Client \( i \) is corrupted. Then \( S_{\text{eff}}^R \) proceeds according to the steps of the protocol as follows.

- **Round 1.** Invoke the adversary on its input \( x_i \), and (common) randomness \( r \), and observes the shares sent by the adversary to the servers in Round 1.

- **Calling the functionality:** Extract from the adversary’s messages an effective input \( x_i^* \) as follows. (1) If there exist sets \( T \) that receive a consistent sharing of \( x_i \), and the sharings of all such sets agree with the same degree-\((t, t)\) polynomial \( Q(z_1, z_2) \), set \( x_i^* = Q(0, 0) \). (2) Else, if all sets \( T \) receive an inconsistent sharing, let \( x_i^* = 0 \). (3) Otherwise, let \( T \) be the first set receiving a consistent sharing of a value \( x_i^* \). Set \( x_i^* = x_i^* \).

- **Round 2.** Simulate the incoming messages of Round 2 as follows. Let \( S \) be the simulator guaranteed by the PSM privacy. Let the PSM outputs of sets receiving consistent shares be \( S(1|x_i^*, out + s_{i,T}) \). For sets \( T \) receiving inconsistent shares output \( S(1|x_i^*, \bot) \) as their Round 2 messages. For each \( T \in \mathcal{T} \), simulate the messages received in MCDS execution \( (i, T) \) using the \( R_T(z_1, z_2), C_T(z_1, z_2) \) induced by the shares and (fresh) randomness \( r_{i,T} \) with \( s_{i,T} \) being its “secret” part.

To prove the “validity” of \( S_{\text{eff}}^R \), we consider the three cases as in \( S_{\text{eff}} \)’s specification.

- If case (1) occurs, then the PSM outputs to Client \( i \) in the real world are exactly as simulated. In particular, sets that received consistent (with \( x_i^* \)) shares, send PSM outputs consistent with \( out + s_{i,T} \) (where \( out = f(x_i^*, x_{[n]\{i\}}) \)), for some random value \( s_{i,T} \). The MCDS outputs are clearly perfectly simulated (jointly with the rest of the view), since we feed them with inputs distributed exactly as in the real execution. The uncorrupted parties receive \( f(x_i^*, x_{[n]\{i\}}) \) as the PSM output of all sets receiving valid shares (there is at least one such set), and a blaming of Client \( i \) otherwise, and thus output \( out \) with probability 1 (as in the ideal world).

- If case (2) occurs, then the PSM outputs of all sets are \( \bot \) (that is, the PSM output messages are statistically simulated by \( S(\ell, \bot) \)). The MCDS replies to Client \( i \) are perfectly simulated, since execution \( (i, T) \) is fed with a random \( s_{i,T} \), and \( R, C \) as shared by Client \( i \). Honest clients output \( f(x_i = 0, x_{[n]\{i\}}) \) in the ideal world by definition of \( S_{\text{eff}}^R \), and in the real world due to the “backup” value received from all sets (while blaming Client \( i \)).
• If case (3) occurs (this is the most interesting case), the PSM outputs of sets $T$ receiving a consistent sharing (from Client $i$) of some value $x_{i,T}^*$ (which may differ between different sets) to Client $i$ are $f(x_{i,T}^*) + s_{i,T}$, where $s_{i,T}$ is a random independent value (which is part of the server shared randomness $r$). Thus, the PSM output itself is a random value, independent of other sets’ PSM outputs. The PSM output to Client $i$ by all other sets is $\bot$. On the other hand, we prove that $R, C$ satisfy the preconditions for MCDS secrecy, so the reply messages of execution $(i, T)$ of MCDS (to Client $i$) are independent of $s_{i,T}$ (even given $R, C$). To simulate those, $\mathcal{S}_\text{hyp-res}$ executes MCDS $(i, T)$ with secret $s_{i,T}$, and $R, C$ as servers’ inputs, but by MCDS secrecy, the replies’ distribution is independent of the concrete secret $s_{i,T}$ (so its joint distribution with the PSM outputs is perfectly simulated). Clearly, uncorrupted clients’ output is $f(x_{i,T}^*, x_{[n] \setminus \{i\}})$, where $T$ is the first set receiving a consistent sharing, and $x_{i,T}^*$ is the induced shared value, both in the real and in the ideal worlds (with probability 1).

It remains to prove that $R, C$ indeed satisfy the MCDS secrecy preconditions. By definition of (3), there exist two sets $T_i \neq T_j$ that each receives consistent sharings of $x_i$, but there is no single degree-$(t, t)$ polynomial both sharings are consistent with. Then, by Schwartz-Zippel, we have at least a $1 - 2t/8t = 3/4$ fraction of disagreement between $R(x, y)$ and $C(x, y)$ on $T_i \times T_j$ (or $T_j \times T_i$). By a standard averaging argument, we conclude that at least $8t \cdot 3/4 \cdot 1/2 = 3t$ of the rows in $T_i \times [n]$ contain at least $3t$ “disagreements” each. Since we need only $2t + 1 \leq 3t$ (for $t \geq 1$) disagreements, we conclude that $R, C$ satisfy these preconditions for $M = T_i \times [n]$.

### 3.4 Full security for $t = 1$

In this section, we return to the standard model where all parties may contribute inputs and receive outputs. We present a 2-round protocol in this model for $n \geq 5$ parties and $t = 1$. This protocol uses some similar ideas to our basic client-server protocol above, but it is different in the types of secret sharing scheme and set system that it employs. Specifically, we use the following ingredients:

1. A 1-private pairwise verifiable secret sharing scheme (see Section 3.2.3). For simplicity, we use here the CNF scheme, though one could use the bivariate version of Shamir’s scheme for better efficiency. Recall that in the 1-private CNF scheme the secret $s$ is shared by first randomly breaking it into $n$ additive parts $s = s_1 + \ldots + s_n$, and then distributing each $s_i$ to all parties except for party $i$. Here we can view a secret as an element of $\mathbb{F}_2^n$.

2. A robust $(n - 2)$-party PSM protocol (see Section 3.2.1). In particular, such a PSM protocol ensures that the effect of any single malicious party on the output can be simulated in the ideal model (allowing the simulator to send “abort” to the functionality).

3. A simple set system, consisting of the $\binom{n}{2}$ sets $T_{i,j} = [n] \setminus \{i, j\}$. (Note that, for $n \geq 5$, we have $|T_{i,j}| \geq 3$.)

Again, we assume for simplicity that members of each set $T_{i,j}$ share common randomness $r_{i,j}$. Similarly to the client-server setting, this assumption can be eliminated by letting 3 of the parties in $T_{i,j}$ pick their candidate for $r_{i,j}$ and distributing it to the parties in the set (in Round 1 of our protocol), and then letting $T_{i,j}$ execute the PSM sub-protocol (in Round 2) using each of the 3 candidates and sending the outputs to
Our protocol proceeds as follows:

- **Round 1:** Each party $P_i$ shares its input $x_k$ among all other parties using a 1-private, $(n-1)$-party CNF scheme (i.e., each party gets $n-2$ out of the $n-1$ additive shares of $x_k$). In addition, to set up the consistency checks, each pair $P_i, P_j$ ($i < j$) generates a shared random pad $s_{i,j}$ by having $P_i$ pick such a pad and send it to $P_j$.

- **Round 2:** For each “dealer” $P_k$, each pair $P_i, P_j$ send the $n-3$ additive shares from $P_k$ they should have in common, masked with the pad $s_{i,j}$, to all parties. Following this stage, each party $P_i$ has an inconsistency graph $G_{i,k}$ corresponding to each dealer $P_k$ ($k \neq i$), with node set $[n] \setminus \{k\}$ and edge $(j,l)$ if $P_j, P_l$ report inconsistent shares from $P_k$.

In addition, each set $T_{i,j}$ invokes a robust PSM protocol whose inputs are all the shares received (in Round 1) by the $n-2$ parties in this set, and whose outputs to $P_i, P_j$ (which are not in $T_{i,j}$) are as follows:
- If all input shares are consistent with some input $x$, then both $P_i, P_j$ receive $v = f(x)$.
- Else, if shares originating from exactly one $P_k$ are inconsistent, then $P_k$ gets ⊥ (in case $k \in \{i, j\}$) and the other party(s) get an “accusation” of $P_k$; namely, a pair $(k, x^*)$ where $x^* = (x_1, \ldots, x_{k-1}, x'_k, x_{k+1}, \ldots, x_n)$. Here, each $x_j$ (for $j \neq k$) is the protocol input recovered from the (consistent) shares and $x'_k = x_k$ if the shares of any $n-3$ out of the $n-2$ parties in $T_{i,j}$ are consistent with each other and $x'_k = 0$ (a default value) otherwise.
- Else, if shares originating from more than one party are inconsistent, output ⊥.

- **Reconstruction:** Each party $P_i$ uses the $n-1$ inconsistency graphs $G_{i,k}$ ($k \neq i$), and the PSM outputs that it received, to compute its final output:
  (a) If some inconsistency graph $G_{i,k}$ has $\text{VC}(G_{i,k}) \geq 2$ then the PSM output of $T_{i,k}$ is of the form $(k, x^*)$; substitute $x_k^*$ by 0, to obtain $x'$, and output $f(x')$.
  Else, (b) if some inconsistency graph $G_{i,k}$ has a vertex cover $\{j\}$ and at least 2 edges, consider the PSM outputs of $T_{i,j}, T_{i,k}$ (assume that $i \neq j$; if $i = j$ it is enough to consider the output of $T_{i,k}$). If any of them outputs $v$ of the form $f(x)$ then output $v$; otherwise, if the output is of the form $(k, x^*)$, output $f(x^*)$.
  Else, (c) if some inconsistency graph $G_{i,k}$ contains exactly one edge $(j,j')$, consider the outputs of $T_{i,j}, T_{i,j'}$ (again, assume $i \notin \{j, j'\}$), and use any of them which is non-⊥ to extract the output (either directly, if the output is of the form $f(x)$, or $f(x^*)$ from an output $(k, x^*)$).
  Finally, (d) if all $G_{i,k}$’s are empty, find some $T_{i,j}$ that outputs $f(x)$ (with no accusation), and output this value.

**Proof Sketch** Intuitively, a corrupted party $P_d$ may deviate from the protocol in very limited ways: it may distribute inconsistent shares (in Round 1) which will be checked (in Round 2) and will either be caught (if

---

17This is similar to Round 2 of the 2-round VSS protocol of [55], except that we use point-to-point communication instead of broadcast; note that, in our case, if the dealer is corrupted, then all other parties are uncorrupted.
the inconsistency graph has VC larger than 1) or will be “corrected” (either to a default value or to its original input, if the VC is of size at most 1). $P_d$ may report false masked shares, for the input of some parties, but this will result in very simple inconsistency graphs (with vertex cover of size 1) that can be detected and fixed. And, finally, $P_d$ may misbehave in the robust PSM sub-protocols (in which it participates) but this has very limited influence on their output (recall that, for sets in which $P_d$ participates, it does not receive the output). We obtain

**Theorem 3.10.** There exists a general, 2-round MPC protocol for $n \geq 5$ parties which is fully secure with straight-line simulation (and guaranteed output delivery) against a single malicious party. The protocol provides statistical security for functionalities in NC$^1$ and computational security for general functionalities by making a black-box use of a pseudorandom generator.

The following section contains a detailed proof of Theorem 3.10.

### 3.4.1 Proof of Theorem 3.10

We describe a (UC) simulator $S$ to simulate the view of a malicious adversary controlling some party $P_d$. As in Section 3.3.2, we will assume that the computationally-unbounded adversary is deterministic; hence, the outgoing messages (in particular, those of Round 2) are determined by the adversary’s input and its incoming messages. The simulator proceeds according to the steps of the protocol, as follows.

- **Round 1:** $S$ feeds the adversary with random shares from the uncorrupted parties. Namely, for each $i$, it feeds the adversary with the $n-2$ random values it expects to get from $P_i$’s CNF-sharing of $x_i$. In addition, it feeds the adversary with a random pad $s_{i,d}$ from each party $P_i$ with $i < d$. This simulates incoming messages of the first round. Then, $S$ observes the shares and the pads sent by the adversary to uncorrupted parties in Round 1.

- **Calling the functionality:** The simulator $S$ extracts from the adversary’s messages an effective input $x'_d$ as follows: it considers the consistency graph $G_{i,d}$, for any $i$ (note that, since only $P_d$ is corrupted, they are all identical). If the graph is empty then all shares are consistent with a unique value $u$ and so $x'_d = u$; if $\text{VC}(G_{i,d}) = 1$ then there is a unique vertex cover of size one $\{j\}$ (the graph cannot contain only a single edge $(j, j')$ since it has at least one more node that must be inconsistent with either $j$ or $j'$) and so a unique value $x'_d = u$ can be reconstructed by ignoring the shares sent to $P_j$. Finally, if $\text{VC}(G_{i,d}) \geq 2$ then $x'_d = 0$.

$S$ calls the functionality with input $x'_d$ and obtains an output $v$.

- **Round 2:** Using the output, $S$ simulates the incoming messages of Round 2. Specifically, (1) for each $i, j, m$ (different than $d$) the masked shares from $P_m$ that $P_i, P_j$ exchange, are simulated by random values; and (2) the PSM messages, received by $P_d$ from the parties in each set $T_{i,d}$, are simulated using the PSM simulator as follows (note that in sets $T_{i,j}$, where $d \notin \{i, j\}$, party $P_d$ receives no message and that it has no access to the PSM randomness in sets $T_{i,d}$): if $G_{i,d}$ is empty then the PSM simulator is given $v$ as the output and produces, in return, the messages sent to $P_d$ (assume, for simplicity, that the PSM-simulator is perfect$^{18}$); if $\text{VC}(G_{i,d}) = 1$ and $\{j\}$ is the unique vertex cover of size 1, then

$^{18}$Actually known robust-PSM protocols guarantee only statistical or computational simulation (depending on the computed function being in NC$^1$ or in POLY, respectively).
the output for the PSM-simulator corresponding to $T_{i,d}$ will be $v$, while for all other $T_{i,d}$ (with $i \neq j$) it is $\bot$. Finally, if $VC(G_{i,d}) \geq 2$ then the PSM-simulator is given $\bot$ as an output (in this case we do not even use the value $v$ that came from the functionality).

The above simulator already shows privacy; namely, for every adversary (controlling one party) and every input for the uncorrupted parties, the output of the simulator is identical (up to the above assumption regarding the robust-PSM simulation). The robustness (and, in particular, correctness) of the protocol is argued next by considering also the (deterministically-computed) messages sent by the adversary to the uncorrupted parties and proving that they do not harm their view (namely, the joint distribution of adversary’s view and uncorrupted parties output is “the same” when the adversary participates in the real protocol or when it is simulated). In particular, we show that every uncorrupted party will output the “correct” output $v$, as above (note that only here we need to consider the sets $T_{i,j}$ where $P_d$ is one of the input parties and not an output party):

If the inconsistency graph $G_{i,d}$, of the malicious $P_d$, has $VC(G_{i,d}) \geq 2$ then $P_i$ knows that $P_d$ is corrupted, and recovers the output $v = f(x')$ using the PSM output received from $T_{i,d}$ (Case (a) of the Reconstruction step).

Otherwise, every graph $G_{i,j}$ has $VC(G_{i,j}) \leq 1$.\(^{19}\) If some graph has at least 2 edges (Case (b)), it is either $G_{i,d}$ with $\{j\}$ as its (unique) vertex cover or $G_{i,j}$ with $\{d\}$ as its (unique) vertex cover. In this case, $P_i$ bases its output on the PSM outputs corresponding to two sets $T_{i,j}$ and $T_{i,d}$, where $P_d$ is corrupted and $P_j$ is uncorrupted ($P_i$ does not know which is which). The set $T_{i,d}$ reports to $P_i$ either the value $f(x)$ itself (if $P_d$ handed the parties in $T_{i,j}$ consistent shares) or the value $(d, x^*)$, with $x^* = x$ (if it handed consistent shares to some $n-3$ of the parties in $T_{i,d}$; this is successfully corrected to $x^* = x$); in both cases $v = f(x)$ is recovered. Any additional inconsistency implies $VC(G_{i,d}) > 1$ which cannot be the case if we passed Case (a). The set $T_{i,j}$ cannot report a conflicting value, since all parties in $T_{i,j} \setminus \{d\}$ hold consistent shares. So, $P_d$ can either force the output to $\bot$ (by reporting inconsistent shares for more than one input or by deviating from the PSM protocol) or else, if it reports inconsistent shares for exactly one input then, again, this input will be successfully corrected.

Suppose that $P_i$ reaches Case (c) and finds some consistency graph has a single edge. It cannot be $G_{i,d}$ (since $n \geq 4$, inconsistent shares from $P_d$ implies that $G_{i,d}$ has at least 2 edges but we already passed Cases (a) and (b)); namely, it must be a graph $G_{i,j'}$ with an edge $(d, j)$ (again, $P_i$ does not know which one is corrupted and which is not). Then, $T_{i,d}$ includes no inconsistent shares (otherwise $G_{i,d}$ would have more than 1 edge) and outputs $f(x)$ (with no accusations). Again, $T_{i,j}$ cannot output any conflicting non-\(\bot\) value, since originally all shares held by $T_{i,j} \setminus \{d\}$ are consistent (and induce $x$). $P_d$ can only provide inconsistent shares for either one or more $x_i$’s (resulting in $x^* = x$ with accusation, or $\bot$ respectively).

Finally, if all graphs are empty, then all inputs were shared properly. Therefore, at least one set has no corrupted parties in it (namely, $T_{i,d}$) and returns the correct output with no accusation.

\(^{19}\)In addition to $G_{i,d}$ it may be that some other graphs have $VC(G_{i,j}) = 1$; in which case $\{d\}$ is a vertex cover for these additional graphs. In some cases (e.g., if there are at least three such graphs) then $P_i$ may conclude which party is corrupted; our protocol does not make use of this information.
3.5 Security with selective abort

This section describes (a simplified, and somewhat weaker, variant) our second main result; namely, a 2-round protocol which achieves security with selective abort against \( t < \frac{n}{3} \) corruptions. This means that the adversary, after learning its own outputs, can selectively decide which uncorrupted parties will receive their (correct) output and which will output "\( \perp \)". More precisely, we prove the following theorem:

**Theorem 3.11.** There exists a general 2-round MPC protocol for \( n > 3t \) parties which is \( t \)-secure, with selective abort. The protocol provides statistical security for functionalities in \( \text{NC}^1 \) and computational security for functionalities in POLY, assuming the existence of a pseudorandom generator in \( \text{NC}^1 \).

A stronger version of this theorem (making a black-box use of an arbitrary PRG) will be given in Section 3.5.4.

The high-level approach for proving Theorem 3.11 is to apply a sequence of reductions, where the end goal is to construct a 2-round protocol that only satisfies the relaxed notion of “privacy with knowledge of outputs” (see Section 2.3), and only applies to vectors of degree-3 polynomials. Concretely:

1. We concentrate, without loss of generality, on functionalities which are deterministic with a public output.

2. We reduce (using unconditional one-time MACs) the secure evaluation of a function \( f \in \text{POLY} \) to a private evaluation, with knowledge of outputs, of a related functionality \( f' \in \text{POLY} \). The reduction is statistical, and if \( f \in \text{NC}^1 \) then so is \( f' \).

3. We reduce the private evaluation with knowledge of outputs of a function \( f' \in \text{POLY} \) to a private evaluation with knowledge of outputs of a related functionality \( f'' \), where \( f'' \) is a vector of degree-3 polynomials. The reduction (using [74]) is perfect for functions in \( \text{NC}^1 \), and only computationally secure (using [4]) for general functionalities in POLY.

4. We present a 2-round protocol that allows \( dt + 1 \) parties to evaluate a vector of degree-\( d \) polynomials, for all \( d \geq 1 \), and provides privacy with knowledge of outputs. In particular, for \( d = 3 \) the protocol requires \( n = 3t + 1 \) parties.

In Section 3.5.1 we provide details on step 4, and in Section 3.5.2 we provide details for step 3. Section 3.5.3 includes more intuition and details, as well as rigorous proofs for the various steps. This includes step 3, which is almost straight-forward, but involves some subtleties that need to be discussed.

### 3.5.1 A private protocol with knowledge of outputs (step 4)

In this section, we present a 2-round protocol for degree-\( d \) polynomials which is private with knowledge of outputs. Let \( p(x_1, \ldots, x_m) \) be a multivariate polynomial over a finite field \( \mathbb{F} \), of total degree \( d \). Assume, without loss of generality, that the degree of each monomial in \( p \) is exactly \( d \).\(^{20}\) Hence, \( p \) can be written as

\[ p(x_1, \ldots, x_m) = \sum_{d' < d} \frac{p(x_1, \ldots, x_m)}{x_0^{d-d'}} x_0^{d-d'} \]

\( \)

\(^{20}\)Otherwise, replace each monomial \( m(x) \) of degree \( d' < d \) by \( m(x) \cdot x_0^{d-d'} \), where \( x_0 \) is a dummy variable whose value is set to 1 (by using some fixed valid \( n \)-tuple of shares).
\[ p = \sum_{g_1 \leq \ldots \leq g_d} \alpha_g \prod_{l=1}^{d} x_{g_l} \]

We start by describing a protocol for evaluating \( p \) with security in the semi-honest model. (This protocol is similar to previous protocols from [14, 73].) The protocol can rely on any \( d \)-multiplicative secret sharing scheme over \( \mathbb{F} \). Recall that, in such a scheme, each party should be able to apply a local computation \text{MULT} to the shares it holds of some \( d \) secrets, to obtain an additive share of their product.

- **Round 1**: Each party \( P_i, i \in [n] \), shares every input \( x_h \) it holds by computing shares \( (s_{i1}^h, \ldots, s_{in}^h) \), using the \( d \)-multiplicative scheme, and distributes them among the parties. \( P_i \) also distributes random additive shares of 0; i.e., it sends to each \( P_j \) a field element \( z_i^j \) such that \( z_i^1, \ldots, z_i^n \) are random subject to the restriction that they sum up to 0.

- **Round 2**: Each party \( P_i, i \in [n] \), computes \( y_i = p_i(s_i^1, \ldots, s_i^m) + \sum_{j=1}^{n} z_i^j \), where \( p_i(s_i^1, \ldots, s_i^m) = \sum_{g_1 \leq \ldots \leq g_d} \alpha_g \text{MULT}(i, s_{i1}^{g_1}, \ldots, s_{in}^{g_d}) \). It sends \( y_i \) to all parties.

- **Outputs**: Each party computes and outputs \( \sum_{i=1}^{n} y_i \) which is equal to \( p(s^1, \ldots, s^m) \), as required.

We will refer to the above protocol as the “basic protocol”. The proof of correctness and privacy in the semi-honest case are standard, and are omitted. Interestingly, this basic protocol happens to be private with knowledge of outputs (but not secure) against malicious parties already for \( d = 2 \), when using Shamir’s scheme as its underlying secret sharing scheme. However, the following simple example demonstrates that the basic protocol is not private against malicious parties already for \( d = 3 \).

**Example 3.1.** Consider 4 parties where only \( P_1 \) is corrupted and the parties want to compute the degree-3 polynomial \( x_1 x_2 x_3 \) (party \( P_1 \) has no input). We argue that, when \( x_3 = 0 \), party \( P_1 \) can compute \( x_2 \), contradicting the privacy requirement. Let \( q_2(z) = r_2z + x_2 \) and \( q_3(z) = r_3z \) be the polynomials used by \( P_2, P_3 \) (respectively) to share their inputs. Their product is \( q(z) = r_2 r_3 z^2 + x_2 r_3 z \). Note that the messages sent by \( P_1 \) to the other 3 parties in Round 1 can make \( P_1 \) learn (in Round 2) an arbitrary linear combination of the values of \( q(z) \) at 3 distinct points. Since the degree of \( p \) is at most 2, this means that \( P_1 \) can also learn an arbitrary linear combination of the coefficients of \( q \). In particular, it can learn \( x_2 r_3 \). This alone suffices to violate the privacy of \( x_2 \), because it can be used to distinguish with high probability between, say, the case where \( x_2 = 0 \) and the case \( x_2 = 1 \).

To prevent badly-formed shares from compromising privacy, we use the following variant of conditional disclosure of secrets (CDS) [59] as a building block. This primitive will allow an honest player to reveal a secret \( s \) subject to the condition that two secret values \( a, b \) held by other two uncorrupted players are equal.

**Definition 3.10.** An MCDS (multiparty CDS) protocol is a protocol for \( n \) parties, which include three distinct special parties \( S, A, B \). The sender \( S \) holds a secret \( s \), and parties \( A, B \) hold inputs \( a, b \) (respectively). The protocol should satisfy the following properties (as usual, the adversary is rushing).

1. If all parties are uncorrupted, and \( a = b \), then all parties output \( s \).

\[21\text{Note that degree-3 polynomials are “complete” in the sense that they can be used to represent every function, whereas degree-2 polynomials are not [73].}\]
2. If \( a = b \), and \( A, B \) are uncorrupted, then the adversary’s view is independent of \( a \), even conditioned on \( s \).

3. If \( a \neq b \), and \( A, B, S \) are uncorrupted, then the adversary’s view is independent of \( s \), even conditioned on \( a, b \).

Note that there is no requirement when \( a \neq b \) and some of the special parties are corrupted (e.g., a corrupted \( A \) may still learn \( s \)). To be useful for our purposes, an MCDS protocol needs to have only two rounds, and also needs to satisfy the technical requirement that the message sent by \( A \) and \( B \) in the first round do not depend on the values \( a \) and \( b \).

Throughout this section, this is the definition of MCDS used (not to confuse with Definition 3.9).

A simple MCDS protocol with the above properties may proceed as follows (see Section 3.5.3 for a proof): In Round 1, party \( A \) picks random independent values \( r, z \in \mathbb{F} \) and sends them to \( B \), and party \( S \) sends \( s \) to \( A \). In Round 2, \( A \) sends to each of the parties \( m_A = a \cdot r - z + s \) and \( B \) sends \( m_B = z - b \cdot r \). Each party outputs \( m_A + m_B \).

An MCDS protocol as above will be used to compile the basic protocol for \( n = dt + 1 \) semi-honest parties into a protocol \( \Pi_{\text{priv}} \) which is private against malicious parties. For this, we instantiate the basic protocol with a \( d \)-multiplicative secret sharing scheme which is also pairwise-verifiable and efficiently extendable (see Section 3.2.3). More precisely, the parties run the basic protocol, and each party \( P_i \) masks its Round 2 message with a sum of random independent masks \( s_{i,j,k,h} \), corresponding to a shared input \( x_h \) and a pair of parties \( P_j, P_k \) (not holding \( x_h \)). In parallel, the MCDS protocol is executed for revealing each pad \( s_{i,j,k,h} \) under the condition that the shares of \( x_h \) given to \( P_j \) and \( P_k \) are consistent, as required by the pairwise-verifiable scheme (where \( a, b \) in the MCDS are values locally computed by \( P_j, P_k \) that should be equal by the corresponding local check). Intuitively, this addresses the problem in Example 3.1 by ensuring that, if a party sends inconsistent shares of one of its inputs to the uncorrupted parties, some consistency check would fail (by pairwise-verifiability), and thus at least one random mask is not “disclosed” to the adversary, and so the adversary learns nothing.

The resulting protocol \( \Pi_{\text{priv}} \) proceeds as follows:

- **Round 1:**
  - Each party \( P_i, i \in [n] \) shares every input \( x_h \) it holds by computing shares \( (s_{i}^{h_1}, \ldots, s_{i}^{h_n}) \) and distributing them among the parties. Each \( P_i \) also sends to each \( P_j \) a share \( z_j^i \) where \( z_1^i, \ldots, z_n^i \) form a random additive sharing of 0.

  - Each triple of distinct parties \( P_i, P_j, P_k \) such that \( j < k \) runs, for each \( h \in [m] \) such that \( x_h \) is not held by \( \{P_i, P_j, P_k\} \), Round 1 of the MCDS protocol (playing the roles of \( S, A, B \) respectively, where all \( n \) parties receive the MCDS output), with secret \( s = s_{i,j,k,h} \), selected independently at random by \( P_i \).

- **Round 2:**
  - Each party \( P_i, i \in [n] \), computes \( y_i = p_i(s_{i}^{1}, \ldots, s_{i}^{m}) + \sum_{j=1}^{n} z_j^i \), where \( p_i(s_{i}^{1}, \ldots, s_{i}^{m}) \triangleq \sum_{g_1,\ldots,g_d \in [m]} \alpha g \text{MULT}(i, s_{i}^{g_1}, \ldots, s_{i}^{g_d}) \). It sends \( y_i' \triangleq y_i + \sum_{j,k,h} s_{i,j,k,h} \) to all parties.
Lemma 3.7. Suppose $\Pi_{\text{priv}}$, described above, computes the degree-$d$ polynomial $p$ and satisfies statistical $t$-privacy with knowledge of outputs. Then the protocol $\Pi_{\text{priv}}$, instantiated with bivariate Shamir, with the above discussion, we get the following lemma:

**Lemma 3.8.** For any $d \geq 1$ and $t \geq 1$, there exists a 2-round protocol for $n = dt + 1$ parties which evaluates a vector of polynomials of total degree $d$ over a finite field $\mathbb{F}$ of size $|\mathbb{F}| \geq n$, such that the protocol is statistically $t$-private with knowledge of outputs.

Our protocols were described as if we need to evaluate a single polynomial. To evaluate a vector of polynomials (which is actually required for our application), we make the following observation. Both the basic semi-honest protocol and $\Pi_{\text{priv}}$ can be directly generalized to this case by running one copy of Round 1, except for the additive shares of 0 that are distributed for each output, and then executing Round 2 separately for each polynomial (using the corresponding additive shares). Observe that for each polynomial in the vector, MCDS is executed for all inputs, rather than only those appearing in that polynomial. The analysis of the extended protocols is essentially the same. Combining $\Pi_{\text{priv}}$, instantiated with bivariate Shamir, with the above discussion, we get the following lemma:

**Lemma 3.9.** Suppose there exists a PRG in $\text{NC}^1$. Then, for any $n$-party functionality $f$, there exists a 2-round MPC protocol which is (computationally) $t$-private with knowledge of outputs, assuming that $n > 3t$. Alternatively, the protocol can provide statistical (and unconditional) privacy with knowledge of outputs for $f \in \text{NC}^1$. In all cases, the simulation is straight-line.
3.5.2 From privacy with knowledge of outputs to security with selective abort (step 2)

The final step in our construction is a reduction from secure evaluation of functions with selective abort to private evaluation with knowledge of outputs. For this, we make use of unconditional MACs. Our transformation starts with a protocol $\Pi'$ for evaluating a single output function $f$, which is private with knowledge of outputs. We then use $\Pi'$ to evaluate an augmented (single output) functionality $f'$, which computes $f$ along with $n$ MACs on the output of $f$, where the $i$-th MAC uses a private key chosen by party $P_i$ at random. That is, $f'$ takes an input $x$, and $k_i \in K$ from each party $P_i$, and returns $y = f(x)$ along with $\text{MAC}(y, k_1), \ldots, \text{MAC}(y, k_n)$. The protocol $\Pi$ is obtained by running $\Pi'$ on $f'$ and having each party $P_i$ locally verify that the output $y$ it gets is consistent with the corresponding MAC. If so, then $P_i$ outputs $y$; otherwise, it outputs $\bot$. Intuitively, this is enough for getting security with selective abort since to make an uncorrupted party output an inconsistent value, the adversary would have to find $y'$ with $\text{MAC}(y', k) = \text{MAC}(y, k)$ for a random unknown $k$ and a known $y$, which can only be done with negligible probability. A formal construction and a proof of Theorem 3.11 appear in Section 3.5.3.6.

Remark 3.3. If broadcast is allowed, then the security notion in Theorem 3.11 can be upgraded to security with abort (that is, achieve agreement). This can be done by defining $f'$ using public-key signatures, rather than using statistical MACs. More specifically, we define

$$f'(x, s k_1, \ldots, s k_n) = (f(x), \text{sig}(s k_1, f), \ldots, \text{sig}(s k_n, f)).$$

The protocol $\Pi$ in this case is obtained by replacing MAC keys $k_i$ by secret keys $s k_i$ for the signature scheme, and letting party $P_i$ broadcast the public key corresponding to $r_i$ in Round 2. Each party then verifies that all $n$ signatures are valid signatures of the output $y = f(x)$ it gets (using the public keys). If so, it outputs $y$, and outputs $\bot$ otherwise. Intuitively, agreement is achieved, since the condition each party verifies in order to decide whether to abort depends on public information (the public keys of other parties). This is unlike in the original protocol, where $P_i$'s condition depends on its MAC key $k_i$, which remain unknown to the other parties. Also, the adversary is unable to make an uncorrupted party output an inconsistent, non-$\bot$ value, since this would imply the ability to forge a signature.

Remark 3.4. It is not hard to generalize the above protocol to be secure with selective abort in a “generalized” client server setting, where the set of clients and the set of servers need not be disjoint (but the communication pattern is only between clients and servers, see [41] for more details on the framework). Note that this setting is a strict generalization of the standard setting, encompassing both it (by letting the set of clients and the set of servers be the same set), and our client-server setting (see abstract for a definition) as special cases. The protocol is secure against any subset of clients, and at most $t < n/3$ servers, for $n > 3$ (hence being a generalization of the protocol above). The construction is essentially the same as the one above, except for the fact that the inputs are shared only among the servers, and that only clients receive Round 2 replies. Naturally, it requires to use MCDS with such a communication pattern, which, requires an MCDS implementation satisfying this communication pattern. This, in turn, will require an adaptation of our definition of MCDS, and a different implementation. See 3.5.3.5 for the definition and implementation of such MCDS.
3.5.3 Technical details for the (simplified) construction

3.5.3.1 A simple private protocol for degree-2 functionalities

We observe that a simple private (against malicious parties) \( n = 2t + 1 \)-party protocol for evaluating any degree-2 polynomial \( p(x) \) can be obtained by using the standard polynomial-based protocols, where each party distributes shares of its input using Shamir (with degree-\( t \) polynomials) as well as shares of 0, and in Round 2 each party applies the polynomial \( p \) to its shares and adds all the shares of 0 to its resulting share. This, in turn, is isomorphic to the basic protocol, from Section 3.5, instantiated with Shamir’s secret sharing.\(^{22}\)

Some intuition. The main observation is that all the adversary can do by sharing a malformed \( s_i(z) \) (i.e., distributing shares in Round 1 that are not consistent with any degree-\( t \) polynomial) is to influence the contribution to the output of terms of the form \( s_i(z) \cdot s_j(z) \) (corresponding to a term \( x_i x_j \) in the polynomial \( p(x) \), where \( x_i \) is the input of \( P_i \), held by the adversary, \( x_j \) is an input held by an uncorrupted party, and \( s_j(z) \) is the corresponding Shamir polynomial). Specifically, such terms can be made to contribute to the output (that is, the free coefficient of the resulting polynomial) a value of the form \( a \cdot x_j + b \), where the distribution of \( a \) and \( b \) is known to the adversary, and \( a \) is the same for all such \( j \). More specifically, \( b \) can be simulated given its Round 1 received messages (shares of \( x_j \)), and the Round 1 messages it sends, and \( a \) depends only on the Round 1 shares sent by \( A \) (which are the same for each term \( x_i \cdot x_j \) evaluated!). Therefore, to simulate \( A \)'s view, it suffices to send \( x_i = a \) to the functionality (a shift the output by \( b \) can obviously be simulated easily and does not harm the privacy).

It is instructive to note that this argument does not hold for degree-3 polynomials, since a malformed \( s_i(z) \) makes each term \( x_i \cdot x_j \cdot x_r \) (where \( x_j, x_r \) are held by uncorrupted parties) contribute to the output a value proportional to \((a \cdot x_j + b_j) \cdot (a \cdot x_r + b_r)\), which is generally not equivalent to \( a' \cdot x_j \cdot x_r \), for any \( a' \).

3.5.3.2 A private protocol for degree-\( d \) polynomials

A simple implementation of MCDS. First, we present an implementation for the MCDS primitive from section 3.5, and prove it has the required properties.

Construction 3.8

- Round 1: \( A \) picks random independent values \( r, z \in \mathbb{F} \), and sends them to \( B \). In addition, \( S \) sends \( s \) to \( A \).

- Round 2: \( A \) sends to each of the parties \( m_A = a \cdot r - z + s \) and \( B \) sends \( m_B = z - b \cdot r \).

- Reconstruction: Each party outputs \( out = m_A + m_B \).

\(^{22}\)There we use an additive sharing of 0, but this can be flipped back and forth by using appropriate Lagrange interpolation coefficients.
Correctness proof sketch. We briefly verify the three properties of MCDS.

1. If (in particular) \( A, B, S \) are uncorrupted, all uncorrupted parties output \( m_A + m_B = (a \cdot r - z + s + z - b \cdot r) = (a - b) \cdot r + s = s \) for any \( r, z \).

2. The adversary receives the messages \( m_a = s + a \cdot r - z + s = s + (a \cdot r - z) \) and \( m_b = -(a \cdot r - z) \). Since \( r, z \) are random independent values unknown to the adversary (\( A, B \) are uncorrupted), then \( a \cdot r - z \) appears as a random value even given \( a (= b) \), (for all \( a \)). Thus, \( m_a, m_b \) appear as a random solution to \( x_1 + x_2 = s \).

3. Even given \( a, b \), we get two linear combinations of \( s, r, z \) that do not span \( s \). Since \( r, z \) are random and independent, \( s \) is uniformly distributed given \( m_a = s + a \cdot r - z, m_b = z - b \cdot r \).

4. Clearly, the protocol is 2-round, and all round-1 messages do not depend on \( a, b \), as required.

Using MCDS. In the following, we provide some intuition on how MCDS is used in \( \Pi_{\text{priv}} \) when evaluating a polynomial \( f(x_1, \ldots, x_m) \) (see Lemma 3.7), and proceed to a formal security proof. Recall that the high level idea is to execute the “basic” protocol with an underlying secret sharing scheme which is pair-wise verifiable, and let each party disclose its Round 2 message under the condition that all good parties received a consistent sharing of each variable using a MCDS as described above. More accurately, each party \( P_i \) will act as a sender once for each pair of other parties \( P_j, P_k \) acting as \( A, B \) and variable \( x_h \) not held by \( P_i, P_j, P_k \) (at most \( \binom{n}{2} \cdot m \) MCDS instances overall). Each MCDS will disclose a random secret \( s_{S,A,B,h} \) under the condition that shares of \( x_h \) which \( A \) and \( B \) should have in common are indeed equal. The message sent by \( S \) in Round 2 of the underlying protocol will be masked with all random secrets \( s_{S,A,B,h} \). The idea is that if the adversary gave an inconsistent shares \( a, b \) of \( x_h \) to some pair of uncorrupted parties \( A, B \) in Round 1, then it will not learn the message sent in Round 2 by any other uncorrupted party \( S \) (by condition 3 of MCDS). On the other hand, we make sure that the number of parties is sufficiently large so that the good parties’ shares determine the shared value (if they are all pair-wise consistent). Also, by condition 2 of MCDS, the adversary cannot use the MCDS protocols to learn any additional information about the shares (“playing” the roles of \( a, b \)) distributed in Round 1 of the final protocol. Finally, condition 1 guarantees that the output is correct when all parties are following the protocol. It is crucial to observe that in the above implementation the values \( a, b \) are used only in Round 2 (indeed, in our application, the values of \( a, b \) in MCDS invocations will not be known to \( A \) and \( B \) during Round 1).

Proof of Lemma 3.7 We present a simulator as required by the privacy notion for any unbounded, (and without loss of generality) deterministic adversary \( A \) corrupting \( t' \leq t \) parties (we slightly abuse notation and denote the set of corrupted parties’ indices by \( A \) as well). In the simulation below, we output the simulated incoming messages received by the adversary as the corresponding incoming protocol message, unless stated otherwise.

- **Round 1**: Simulate adversary’s Round 1 incoming messages.
  - For each \( x_h \) held by \( [n] \setminus A \), sample the shares of \( x_h \) received by \( A \) as (say) \( D(0, r)[n] \setminus A \).
For $j \in [n] \setminus A$, $i \in A$ let the 0-shares, $z^j_i$, received by $P_i$ from $P_j$ be a random independent value.

Simulate the incoming Round 1 messages for the $(i, j, k, h)$-th MCDS, independently for each instance. Messages from parties with no special role are simulated as specified by the MCDS, on a random independent input string. Messages from $A, B$ are simulated in the same manner, substituting $a = 0/b = 0$ as their inputs. The Round 1 messages from $S$ are simulated on a random, independent input $s_{i,j,k,h}$.

**Input submission:** Invoke $A$ on its Round 1 messages given its input, and the incoming Round 1 messages as generated above (recall that $A$ is rushing). There are two possibilities:

- (1) For every variable $x_h$ held by $A$, the corresponding effective shares sent to $[n] \setminus A$ are consistent with some value $x^*_h$. In this case, submit the corresponding value sequence $x^*_h$.
- (2) Otherwise, submit 0.

**Round 2:** Simulate $A$’s Round 2 view. Complete the Round 2 incoming messages in all MCDS instances, by simulating the messages from uncorrupted parties on the following inputs and Round 1 incoming messages:

- The randomness used by each party as determined in Round 1.
- $s$ is as determined in Round 1.
- If $x_h$ is held by $A$, use the corresponding (partial) share of $x_h$ sent to $A (B)$ as $a (b)$. In particular, $a, b$ may differ.
- If $x_h$ is not held by $A$, if $A \in A (B \in A)$, set $A$’s ($B$’s) input $a (b)$ to the be this share. Otherwise, leave $A$’s and $B$’s inputs as fixed in Round 1 ($a = b = 0$).
- The Round 1 messages from $A$ are as generated in Round 1 (in the simulation above). The Round 1 messages from other uncorrupted parties are as simulated on their inputs and randomness generated in Round 1.

- If case (2) above happened, let $y_i'$ for each $i \in [n] \setminus A$ be a random value, independent of all values seen before.
- Otherwise (case (1) happened). Let $out$ denote the ideal functionality’s output.

- Let $S = \sum_{i \in [n] \setminus A, j \in [n] \setminus A} s_{i,j,k,h}$, where the $s_{i,j,k,h}$’s are as generated above. Also, let $Z = -\sum_{i \in A, j \in [n] \setminus A} z^j_i + \sum_{i \in [n] \setminus A, j \in A} z^j_i$.
- Complete the sharing for each $x_h$ held by $A$ into a valid sharing $s^h_{1,1}, \ldots, s^h_{n,1}$ of the effective scheme to include shares for parties in $A$.

---

23 This works, since Round 1 messages are independent of $a, b$.

24 Recall that the secret sharing scheme we use is such that this condition can be efficiently verified, and if satisfied, a completion of the partial sharing can be found efficiently. For instance, all LSSS schemes have this property.
For each \(i \in \mathcal{A}\), use the simulated Round 1 incoming messages of sharings of \(x_\mathcal{A}'\)’s held by uncorrupted parties, and the shares generated above to generate an additive share, \(y_i\), of \(p(x_1, \ldots, x_m)\), as specified by the protocol. Complete the \(y_i\)’s into an additive sharing of \(\text{out}\) (for all \(i \in [n]\)). Let the \(y_i'\)’s for \(i \in [n] \setminus \mathcal{A}\) be a random additive sharing of \(V = \text{out} + Z + S - \sum_{i \in \mathcal{A}} y_i\).

- **Submitting uncorrupted parties’ outputs:** Further invoke \(\mathcal{A}\) to generate its Round 2 messages, feeding it the uncorrupted parties’ round-2 messages as simulated above. For each \(i \in [n] \setminus \mathcal{A}\), recover \(P_i\)’s round-2 messages using the simulated incoming and outgoing round-2 messages of \(A\), and compute \(P_i\)’s output \(\text{out}_i\) as specified by the protocol (this is possible since an uncorrupted party’s output depends only on its round-2 messages, which are, in turn, identical for all parties). Submit \(\text{out}_i\) to the functionality as an output to \(P_i\).

As mentioned above, knowledge of outputs follows from the fact that a party’s output depends only on its incoming Round 2 messages, and these are the same to all parties, so the Round 2 messages of other parties, and thus their outputs can be recovered from \(\mathcal{A}\)’s simulated view. It remains to prove that the adversary’s view is perfectly simulated. We first prove that the adversary’s view besides the \(y_i'\)’s is perfectly simulated. Clearly, all Round 1 messages besides the MCDS messages are perfectly simulated (and are independent of the MCDS messages in Round 1). The MCDS messages are also perfectly simulated:

- **(Round 1)** Messages from \(A, B\) are perfectly simulated, since the real-world Round 1 messages do not depend on \(a (b)\), so simulating \(A (B)\) on \(a = b = 0\) perfectly simulates incoming round-1 messages from (uncorrupted) \(A, B\). Messages from \(S\) or non-special parties are perfectly simulated, since its MCDS input is distributed exactly as in the real execution. Messages among uncorrupted parties are thus also perfectly simulated for the same reason.

- **(Round 2)** To see that we obtain perfect simulation (of MCDS messages in both rounds), it is sufficient to observe that the resulting MCDS view of \(\mathcal{A}\) where \(A, B\) in Round 2 are simulated on inputs \(a, b\) as in the simulator description distributed exactly as in the real execution. This is so since the inputs to \(S\) and the uncorrupted parties are distributed as in the real world anyway. In particular, the Round 2 messages are consistent with the Round 1 messages (including messages among uncorrupted parties), since Round 1 messages do not depend on \(a, b\).

This completes the proof that the adversary’s view besides the \(y_i'\)’s is perfectly simulated. The main observation is that if all effective shares sent by \(\mathcal{A}\) to \([n] \setminus \mathcal{A}\) are consistent with some sequence \(x_\mathcal{A}'\), then in the real execution \(\mathcal{A}\) recovers all \(s_{i,j,k,h}\)’s, and the \(y_i'\)’s received from \(i \in [n] \setminus \mathcal{A}\) are indeed distributed as a random sequence summing to \(\text{out} + V = f(x_\mathcal{A}', x_{[n] \setminus \mathcal{A}}) + V\), where \(V\) is determined by the rest of its view as described in the simulator. Otherwise, inconsistent shares of \(x_h\) for some \(h\) have been sent to \(P_j, P_k\), where \(j, k \in [n] \setminus \mathcal{A}\). Thus, by property 3 of the MCDS, for all \(i \in [n] \setminus \mathcal{A} \cup \{j, k\}\), \(\mathcal{A}\) has no information about \(s_{i,j,k,h}\) actually used for masking in round 2. Since there are at least \(n - t \geq 3t + 1 - t = 2t + 1 \geq 3\) uncorrupted parties, the output share of at least \(3 - 2 = 1\) uncorrupted party remains unknown to the adversary. So, because of the \(z_i'\)’s, the \(y_i'\)’s appear as a random sequence summing up to a random value, that is, a random sequence.

We only remark that the above simulator can be generalized to be a simulator for evaluating a vector of degree-\(d\) polynomials in the natural way. In particular, we note the following.
Observation 3.6. In the “natural” generalization of the simulator above to a sequence $V$ of degree-$d$ polynomials, the simulated incoming messages “related” to a subsequence $V'$ (shares, round-2 CDS messages, and “masked” output share) in the vector is independent of the TP’s output for polynomials outside of $V'$ for all subsequences $V'$.

3.5.3.3 Unconditional MACs

In this section we recall the definition and a basic construction of unconditional MACs.

Definition 3.11. An unconditionally $\epsilon$-secure message authentication code (MAC) scheme consists of a pair of deterministic algorithms $\text{MAC}(K, M), \text{Ver}(K, M, \sigma)$, and corresponding domains $K, M$. It satisfies:

- Correctness: for any $K \in K, M \in M$ and $\sigma \leftarrow \text{MAC}(K, M)$, we have $\text{Ver}(K, M, \sigma) = 1$.
- Integrity: for any $M \in M$, and any algorithm $A(\cdot, \cdot)$ (possibly unbounded),
  $$\Pr_{K \in K}[\sigma \leftarrow \text{MAC}(K, M), A(M, \sigma = \text{MAC}(K, M)) = ((M', \sigma') \neq (M, \sigma)) | \text{Ver}(K, M', \sigma') = 1] \leq \epsilon.$$  

A simple instantiation of MACs that is good for our purposes is:

Construction 3.9. $M = \mathbb{F}$, where $\mathbb{F}$ is a field of size $\geq 1/\epsilon$, and $K = \mathbb{F} \times \mathbb{F}$. The scheme is defined by $\text{MAC}(K = (a, b), M) = a \cdot M + b$ and $\text{Ver}(K = (a, b), M, \sigma) = 1$ iff $a \cdot M + b = \sigma$.

The advantage of this construction is that if $M$ itself is of degree $d$ then $\text{MAC}(K, M)$ is of degree $d + 1$. This will be used below.

3.5.3.4 Moving from degree-3 polynomials to general functions (step 3)

In the following, we provide details on the reduction from evaluating general $f \in \text{POLY}$ with privacy and knowledge of outputs to evaluating vectors of degree-3 polynomials with the same security notion [73, 74, 4]. It is essentially an easy adaptation of the standard reduction from the literature. This reduction is rather oblivious to the concrete form of security, and in particular, works for privacy with knowledge of outputs as well. However, there is a subtle issue regarding using such reductions for our needs (in Section 3.5) that we address here. Namely, we represent $f$ by RP as described in Section 2.2 (either statistically or computationally private).

How do we use Randomizing polynomials in our reduction? It is proved in [74] that any function $f : \{0, 1\}^m \rightarrow \{0, 1\}$ can be represented by perfectly private degree-3 polynomials over any field $\mathbb{F}$ of prime size. Furthermore, RP size is polynomial in $m$ for $f \in \text{NC}^1$. In [4], they obtain efficient degree-3 computationally private RP for all $f \in \text{POLY}$, using a PRG in $\text{NC}^1$ (in a non-black-box manner). Our first step is to apply one of these reductions, to obtain a degree-3 RP for $f$.

Now, given a randomized encoding $f'(x_1, \ldots, x_m, r_1, \ldots, r_m') : \mathbb{F}^{m+m'} \rightarrow \mathbb{F}'$ of $f$ via a vector of degree-3 polynomials, we use a standard reduction from secure evaluation of a randomized functionality to secure evaluation of a related deterministic
functionality \( f' \). The reduction lets party \( i \) to pick a value \( r_{i,j} \) of each \( j \in [m'] \) independently at random, and jointly evaluate a function

\[
f''(x_1, \ldots, x_m, r_{1,1}, \ldots, r_{m,m'}) = f'(x, \sum_{i \in [m]} r_{i,1}, \ldots, \sum_{i \in [m]} r_{i,m'}).
\]

Party \( i \) supplies the input bits it holds, and \( r_{i,1}, \ldots, r_{i,m'} \). The overall reduction to secure evaluation of degree-3 polynomials is applicable to the malicious setting (which is what we need), except for the following subtle issue. The adversary may submit values of \( x_j \)'s which are not 0 or 1 (for fields which are larger than \( \mathbb{F}_2 \)), in which case the adversary (and the uncorrupted parties) learn a value which is not necessarily consistent with some input \( x^* \in \{0, 1\}^m \) to \( f \). On the other hand, the protocol for evaluating (vectors of) degree-3 polynomials we construct in Section 3.5.1 works only over large (\(|\mathbb{F}| > n\)) fields if we use bivariate Shamir as the underlying secret sharing scheme (it is convenient to use this scheme since it is 3-multiplicative and pairwise-verifiable, as we need, and its share complexity is polynomial in \( n \)).

**Handling the problem.** We propose two solutions to the problem. The first one alters the construction of randomizing polynomials, to be “meaningful” for values \( x \notin \{0, 1\} \), and uses it to evaluate an extension of \( f \) to \( \tilde{f} : \mathbb{F}^m \to \mathbb{F} \), such that \( \tilde{f}(x) = \tilde{f}(g(x)) \), for some \( g : \mathbb{F}^m \to \{0, 1\}^m \) which is the identity function when restricted to \( \{0, 1\} \). The second approach is to construct a secret sharing over \( \mathbb{F}_2 \) with the properties we need.

**First solution.** We introduce a notion of function extensions.

**Definition 3.12.** The canonic extension of a function \( f : \{0, 1\}^m \to \{0, 1\} \) over a finite field \( \mathbb{F} \) is defined as the function \( \tilde{f}_\mathbb{F} : \mathbb{F}^m \to \mathbb{F} \) such that \( \tilde{f}(x_1, \ldots, x_m) = f(x_1^{\mathbb{F} - 1}, \ldots, x_m^{\mathbb{F} - 1}) \) (where 0, 1 are interpreted as field elements).

Observe that \( \tilde{f}(x) = f(x) \) for all \( x \in \mathbb{F}^m \), and \( y^{\mathbb{F} - 1} = 1 \) if \( y \neq 0 \) and 0 otherwise.

We will use a stronger version of the randomized encodings described above, which is “meaningful” for inputs \( x \in \mathbb{F}^m \) [40]. More precisely.

**Lemma 3.10.** For any finite field \( \mathbb{F} \), and any arithmetic branching program of size \( s \) (as defined in [40]) evaluating a function \( f : \mathbb{F}^m \to \mathbb{F} \), there exists a statistically private randomized encoding \( f' \) of \( f \) via \( l = \text{poly}(s) \) degree-3 polynomials over \( \mathbb{F} \) (in particular, correctness and privacy of the encoding hold for all \( x \in \mathbb{F}^m \), rather than just for binary inputs).

Another fact we need is:

**Fact 3.1.** For \( f \in \text{NC}^1 \), and any finite field \( \mathbb{F} \), there exists an arithmetic branching program computing \( \tilde{f}_\mathbb{F} \) of size \( \text{poly}(m, |\mathbb{F}|) \).

Now, consider a function \( f : \{0, 1\}^m \to \{0, 1\} \in \text{NC}^1 \). We reduce it to the evaluation of a vector of degree-3 polynomials over a “large” field \( \mathbb{F}, |\mathbb{F}| > n \) as we need, as follows. (1) Devise an arithmetic branching program for evaluating \( \tilde{f}_\mathbb{F} \) of size \( \text{poly}(m, |\mathbb{F}|) \). Such a representation is guaranteed to exist by Fact 3.1. (2) Apply Lemma 3.10 to the branching program, to obtain a randomized encoding \( f' \). This
encoding is useful for us, since the distribution on the output of \( f' \) an adversary can induce is consistent with a distribution on some binary input (depending on the input it submitted), by definition of the canonic function. For functions \( f \in \text{POLY} \), we reduce to evaluating a vector of degree-3 polynomials by first applying [4] to obtain a computational encoding of \( f \) via an \( \text{NC}^0 \) function \( f' \), as mentioned above, and then apply the above encoding to \( f' \), to obtain an efficient computational encoding \( f'' \) of \( f \) via randomizing polynomials.

**Second solution.** Although the above transformation keeps the complexity of the resulting protocol polynomial, its concrete overhead can be substantial. Another approach, leading to a more efficient solution is to use a secret sharing scheme over \( \mathbb{F}_2 \) with the properties we need. It turns out that a simple modification of Shamir can be used. Given an \((n = 3t + 1, t)\) Shamir scheme over \( \mathbb{F} = \mathbb{F}_2^l \), we simply limit the range of secrets to be shared to \( \{0, 1\} \), and treat every share that a party previously received as \( l \) shares over \( \mathbb{F}_2 \). This results in a 3-multiplicative \((3t + 1, t)\)-secret sharing scheme \((D, E)\) over \( \mathbb{F}_2 \). We observe that this scheme is an LSSS over \( \mathbb{F}_2 \) (see Section 3.2.3). From Theorem 3.2, we conclude that there exists a pairwise verifiable scheme \((D, E)\) in which the above scheme is embedded. The latter is also 3-multiplicative \((3t + 1, t)\) scheme.

### 3.5.3.5 MCDS for the (generalized) client-server setting

As mentioned in Remark 3.4, the construction from Section 3.5 can be easily modified to work in a (generalized) client-server setting (strictly generalizing it). More specifically, the generalized protocol merely modifies the construction from 3.5 so that clients share their inputs among servers, and only clients receive Round 2 replies (otherwise, we just use the protocol described there as is). The only nontrivial modification is using a client-server variant of MCDS, which only allows communication between clients and servers, and only clients learn the inputs. We will need a different implementation of this primitive (for instance, Construction 3.8 does not work if the set of clients and the set of servers are disjoint, since than the protocol instructs servers to exchange messages in Round 1). We start with formally defining a generalized MCDS.

**Definition 3.13.** A generalized MCDS is defined like MCDS, where additionally the set of parties, \( \text{Part} \), satisfies \( \text{Part} = \text{Clients} \bigcup \text{Servers} \) (the two sets are not necessarily disjoint), where \( A, B, S \in \text{Servers} \). Also, only clients eventually output a value. Additionally, communication is allowed only between clients and servers. The MCDS requirements are generalized as follows.

1. If all parties are uncorrupted, and \( a = b \), then all clients output \( s \).
2. If \( a = b \), and \( A, B \) and some client are uncorrupted, then the adversary’s view is statistically independent of \( a \), even conditioned on \( s \).
3. If \( a \neq b \), and \( A, B, S \) and some client are uncorrupted, then the adversary’s view is statistically independent of \( s \), even conditioned on \( a, b \).

\(^{25}\text{Some of the requirements have a statistical, rather than a perfect flavor, since it is sufficient for our purposes, and since this is the best we could implement.}\)
We additionally make the technical requirements that the MCDS protocol has two rounds, and that the message sent by $A$ and $B$ in the first round do not depend on the values $a$ and $b$.

Next, we present an implementation meeting all requirements.

**Construction 3.10.**

- Let $\mathbb{F}$ denote a large finite field ($|\mathbb{F}| = exp(k)$, for $k$ a security parameter).
- Round 1: Each client $P_i$ picks random independent values $r_{i,1}, r_{i,2}, r_{i,3}, z_{i,1}, z_{i,2} \in \mathbb{F}$, and sends $z_{i,1}$ to $S$, $z_{i,2}, r_{i,1}, r_{i,2}, r_{i,3}$ to $A$, and $z_{i,1}, z_{i,2}, r_{i,1}, r_{i,2}, r_{i,3}$ to $B$.
- Round 2: Denote by $Z_1 = \sum_{i \in \text{Clients}} z_{i,1}$ (similarly, $Z_2, Z_3, R_1, R_2, R_3$).
  - $S$ sends $s + Z_1$ to each client.
  - $A$ sends $(a + R_1 \cdot R_2) \cdot R_3 + Z_2$ to each client.
  - $B$ sends $(b + R_1 \cdot R_2) \cdot R_3 + Z_1 + Z_2$ to each client.
- Reconstruction: Each party outputs $out = m_S + m_A - m_B$.

**Correctness proof sketch.** We verify the three properties of MCDS.

1. If all parties are uncorrupted, and $a = b$, all clients output $m_S + m_A - m_B = s + Z_1 + (a + R_1 \cdot R_2) \cdot R_3 + Z_2 - (b + R_1 \cdot R_2) \cdot R_3 - Z_1 - Z_2 = s + (a - b) \cdot R_3 = s$.

2. If $a = b$, $A, B$ are uncorrupted, and some client is uncorrupted, than wlog. the adversary’s view is equivalent to that in a game where it is given:

   - $m_S = s + Z'_1$
   - $m_A = (a + R'_1 \cdot R'_2) \cdot R'_3 + Z'_2$
   - $m_B = (a + (R'_1 + d_1) \cdot (R'_2 + d_2)) \cdot (R'_3 + d_3) + Z'_1 + Z'_2$

where $Z'_1, Z'_2, R'_1, R'_2, R'_3$ are picked independently at random from $\mathbb{F}$, and $d_1, d_2, d_3, s$ are constants known to the adversary. This is readily seen, recalling that at least one client is uncorrupted, and that the adversary may also participate in picking the randomness $Z_1, Z_2$ etc, and neglecting some known additive constants. We prove that for any values $a_0, a_1$ of $a$, the adversary’s views in the corresponding games are statistically close. $m_S$ is clearly independent of $a$. $m_A$ is distributed uniformly and independently of $m_S$ (due to $Z'_2$). $m_B = (a + (R'_1 + d_1) \cdot (R'_2 + d_2)) \cdot (R'_3 + d_3) + Z'_2 + m_S - s$. Let $D = d_1 \cdot R'_2 + d_2 \cdot R'_1 + d_1 \cdot d_2$, and rewrite $m_B = m_A + m'_3 + D + d_3 \cdot (a + R'_1 \cdot R'_2) + m_S - s$. Consider the event $R'_1 \cdot R'_2 \neq 0$, which occurs with all but a negligible probability ($\leq 2/|\mathbb{F}|$) (so it suffices to focus on it). Also, the conditional distribution is uniform over $\mathbb{F} \setminus \{0\}$, and thus statistically close to uniform on $\mathbb{F}$.(*) There are two cases.

---

(*Note also, that MCDS is a special case where all parties are labeled both as clients and as servers.)
• Assume \( d_1 = d_2 = 0 \). Then \( m_B - m_A = d_3 \cdot (a + R'_1 \cdot R'_2) \). If \( d_3 = 0 \), then \( m_B - m_A = 0 \), so \( m_B = m_A + m_S - s \), and overall the distribution is \((m_S, m_A, m_A + m_S - s)\) is independent of \( a \).

Otherwise, since \( R'_1 \cdot R'_2 \) is uniform over \( \mathbb{F} \setminus \{0\} \), \( m_B - m_A \) is statistically close to being uniform (over \( \mathbb{F} \)) and independent of \( m_A \) (thus so is \( m_B \)). We conclude \((m_S, m_A, m_B)\) is statistically close to a triple of random independent variables.

• Otherwise \((d_1 \neq 0 \) or \( d_2 \neq 0) \), condition on \( c = R'_1 \cdot R'_2 \neq 0 \). For any fixed value of \( R'_1 \), the probability that \( R'_2 \) is picked so that \( D = 0 \) is then \( \leq \frac{1}{|\mathbb{F}|^2} \), making \( m_B - m_A \) statistically close to uniform, and independent of \( m_A \) (due to \( R'_2 \)). We conclude that \((m_S, m_A, m_B)\) is statistically close to a triple of random independent variables.

3. If \( A, B, S \) and some client are uncorrupted and \( a \neq b \), than wlog. the adversary’s view is equivalent to that in a game where

- \( m_S = s + Z'_1 \)
- \( m_A = (a + R'_1 \cdot R'_2 \cdot R'_3) + Z'_2 \)
- \( m_B = (b + (R'_1 + d_1) \cdot (R'_2 + d_2)) \cdot (R'_3 + d_3) + Z'_1 + Z'_2 \)

\( Z'_1, Z'_2, R'_1, R'_2 \) are picked independently at random from \( \mathbb{F} \), and the constants \( d_1, d_2, d_3 \), and \( a \neq b \) are known to the adversary. We prove that for each pair of values \( s_1, s_2 \), the adversary’s views in the corresponding games are statistically close. \( m_S \) is randomly distributed (due to \( Z'_1 \) ). \( m_A \) is random and independent of \( m_S \). Denote \( D = b - a + R'_1 d_2 + R'_2 d_1 + d_1 d_2 \). We have \( m_B = m_A + m_S + D \cdot (d_3 + R'_3) + d_3 \cdot (a + R'_1 \cdot R'_2) - s \). The main observation is that \( D \) is non-0 with overwhelming probability(\( * \)). Once we prove that, we deduce that even conditioned on \((m_S, m_A)\), \( m_B \) is uniformly distributed (then \((m_S, m_A, m_B)\) is statistically close to a random triple of independent variables, and we are done). This, in turn, holds since \( D \cdot (d_3 + R'_3) \) contributes a statistically close to random value (by(\( * \) from the previous item), independent of \( m_A, m_B \) (due to \( Z'_1, Z'_2 \)). It remains to prove (\( ** \)):

- If \( d_1 = d_2 = 0 \), then \( D = b - a \neq 0 \).
- Otherwise, \( D \) is an affine combination of \( R'_1, R'_2 \), where at least one deg-1 coefficient is non-0. Thus it is non-0 with overwhelming probability.

\( \square \)

3.5.3.6 More details for step 3

In the following, we formally present a reduction from evaluating a function \( f \in \text{POLY} \) with selective abort, to (statistically/computationally) evaluating a related functionality \( f' \in \text{POLY} \) with privacy and knowledge of outputs (with the same resilience parameter \( t \)). More formally, we prove the following.

**Lemma 3.11.** Assume that for some \( n, t \), and a function \( f(x_1, \ldots, x_m) \in \text{POLY} \), there exists an \( n \)-party 2-round protocol for evaluating \( f \) with privacy and knowledge of outputs, secure against a \( t \)-adversary. The security is statistical for \( f \in \text{NC}^1 \), and computational (under some assumption) otherwise. Then there exists an \( n \)-party 2-round protocol for evaluating any function \( f(x_1, \ldots, x_m) \in \text{POLY} \) with security with
statistical abort, secure against a $t$-adversary. The security is statistical for $f \in \text{NC}^1$, and computational (under the same assumption) otherwise.

We present a proof for the statistical case, while the proof for the computational case is similar, and is omitted. The proof is by constructing a statistically secure with abort protocol for $f$, using a statistically private with knowledge of outputs protocol for a related function $f'$.

- Let $f(x_1, \ldots, x_n) : \{0, 1\}^l \rightarrow \{0, 1\}$ be a function in POLY. Let $f'((x_1, k_1), \ldots, (x_n, k_n)) = (f(x_1), \text{MAC}(f(x), k_1), \ldots, \text{MAC}(f(x), k_n))$ be an $n$-party functionality, where all parties receive the entire output. Let $\Pi_{f'}$ be an $n$-party 2-round protocol, evaluating $f'$ statistically (computationally) $t$-privately with knowledge of outputs, as guaranteed in the theorem’s condition.\footnote{In the \text{MAC} function, the length of the $k_i$’s is an (implicit) security parameter to the \text{MAC}.}

- Rounds 1.2: Party $P_i$ sets its augmented input to be $y_i = (x_i, k_i)$, and $k_i$ is a MAC key picked independently at random. The parties execute $\Pi_{f'}$ where party $P_i$ has input $y_i$.

- Reconstruction: Party $P_i$ proceeds as follows. Let $\text{out}_i = \langle \text{out}_{i,0}, \text{out}_{i,1}, \ldots, \text{out}_{i,n} \rangle$ denote its output in $\Pi_{f'}$. If $\text{out}_{i,i} = \text{MAC}(\text{out}_{i,0}, k_i)$, output $\text{out}_{i,0}$. Otherwise output $\bot$.

**Security proof sketch.** We present a statistical simulator for any (deterministic, unbounded) $t$-adversary $A$, corrupting a non-empty set of players (if the set is empty, non-triviality of $\Pi_{f'}$ follows by non-triviality of $\Pi'$). Let $S'$ denote a simulator for $A$ (viewed as an adversary attacking $\Pi_{f'}$) guaranteed by $\Pi_{f'}$’s security notion.

- **Calling the functionality.** Give $S'$ oracle access to $A$ (on inputs $x_A$). $S'$ invokes $A$ to obtain the input $y'^*_i = \langle x'^*_i, k'^*_i \rangle_{i \in A}$ submitted to the functionality for $f'$ in the ideal world. On behalf of each corrupted $P_i$, submit $x'^*_i$ to the functionality (computing $f$).

- **Simulating $A$’s view.** Let $\text{out} = f(x_A, x_{[n]\setminus A})$ be the received reply. Let $\text{out}' = \langle \text{out}'_0 = \text{out}, \text{out}'_i = \text{MAC}(\text{out}, k_i), \ldots, \text{out}'_n = \text{MAC}(\text{out}, k_n) \rangle$ where $k_i$ is picked independently at random for all $i \in [n] \setminus A$, and $k_i = k_i^*$ otherwise. Feed $S'$ with $\text{out}'$ as the output from the functionality (for $f'$), to obtain $A$’s view and the outputs $\{\text{out}'_i\}_{i \in [n]\setminus A}$ sent to the functionality (for $f'$) to be submitted to the uncorrupted parties. Output the view output by $S'$ as $A$’s view.

- **Deciding uncorrupted parties’ outputs.** For $i \in [n] \setminus A$, if $\text{out}_{i,i}^* = \text{MAC}(\text{out}_{i,0}^*, k_i)$, instruct the (main) functionality to send $P_i$ the output $\text{out}$. Otherwise, instruct the functionality to send $\bot$ to $P_i$.

We now prove that the above simulator statistically simulates the joint output of the adversary and the uncorrupted parties with selective abort, assuming $\Pi_{f'}$ is a protocol that computes $f'$ with statistical privacy and knowledge of outputs. Privacy follows by privacy (with knowledge of outputs) of $\Pi_{f'}$. That is, the view of $A$ as an adversary attacking $\Pi_f$ is equal to $A$’s view as an adversary attacking $\Pi_{f'}$, and is statistically simulated by $S'$. In turn, the adversary’s view output by $S$ is distributed identically to the adversary’s view output by $S'$ for the same uncorrupted parties’ inputs by construction (in particular, given the inputs $S'$ submits (to $f'$), the functionality’s reply is perfectly simulated). Next, we prove that conditioned on the adversary’s view, the uncorrupted parties’ outputs are statistically simulated. In both real and the ideal
worlds, the output of an uncorrupted party $P_i$ depends only on the output of $\Pi'_f$ it recovers: from the execution of $\Pi_f$ in the former case (denoted $out_i$), and from the output of $S'$ in the latter case (denoted $out'_i$). By knowledge of outputs, these, jointly with the adversary’s view, are statistically close\(^{(5)}\). Now, in the real world, an uncorrupted party $P_i$ outputs a non-$\perp$ value $out_0$, if $out_{i,0} = MAC(out_{i,0}, k_i)$, and $\perp$ otherwise. In the ideal world, its output, assuming $out_{i,0}^* = MAC(out_{i,0}^*, k_i)$ is $out'_0 = f(x_A^*, x_{\overline{A}})$, and $\perp$ otherwise. It remains to prove that in the former case $out_{i,0}^* \neq out'_0$ with negligible (in the $MAC$’s security parameter) probability. This completes the proof because in this case $out_i \neq out_i^*$, with negligible probability (by (5)).

Assume for contradiction that this is not the case, and that for some input $x_{\overline{A}}$, $out_{i,0}^* \neq out'_0$ for some $i \in [n]$ with non-negligible probability for an infinite number of $k$’s (since $n$ is fixed, we can assume for simplicity that it is the same $i$ for all values of $k$). Thus, we can use the simulator $S'$ to break the $MAC$. More concretely, the following procedure breaks the $MAC$.

- Consider an oracle to a random independent $MAC$ functions $g(x) = MAC(x, k_g)$, where $k_g$ is a randomly selected $MAC$ key.
- Run $S'$ given access $A$ (viewed as an adversary attacking $\Pi_f$) on inputs $x_A$, up to the stage it produces the $y_i$’s for $i \in A$. Let $M = f(x_A^*, x_{\overline{A}})$.
- Produce $out' = M, mac_1 = MAC(M, k_1), \ldots, mac_n = MAC(M, k_n)$ by letting $k_j = k_i^*$ for $j \in A$, calling $g_i$ for $j = i$. Otherwise ($j \in [n] \setminus A$), let $mac_j = MAC(M, k_j)$ for a randomly selected $k_j$ otherwise.
- Feed $out'$ to $S'$ to produce the outputs to give to uncorrupted parties. By the contradiction assumption, with non-negligible probability, $out_{i,0}^* = g(out_{i,0}^*)$, for some $out_{i,0}^* \neq M$.

Clearly, the distribution of $out_{i,0}^*$ is exactly as produced by $S'$, so we have managed to generate a second preimage for $g(M)$, where $g$ is a random $MAC$ function corresponding to the security parameter $k$, for an infinite number of $k$’s. We observe that for our specific implementation of $MAC$’s (for a security parameter $k = l$), if $f \in POLY$, then so is $f'$, and if $f \in NC^1$ then so is $f'$, which completes the proof of Theorem 3.11.

Theorem 3.11 follows by combining Lemma 3.11 with the protocol from Lemma 3.8.

### 3.5.4 Making black-box use of arbitrary PRG

The above construction relies on the existence of a PRG in $NC^1$, and makes non black-box use of such a PRG. In the following, we show how to transform the construction to make black-box use of an arbitrary PRG, with the same parameters (round complexity and security threshold). The part of the construction that poses extra requirements on the PRG is the reduction from securely evaluating an arbitrary $f \in POLY$, to securely evaluating (a vector of) degree-3 polynomials. In a nutshell, we rely on a transformation from [4], which interprets Yao’s garbled circuit (applied to an input $x_1, \ldots, x_l$) as a computationally private randomized encoding of the function $f(x_1, \ldots, x_n)$, by a function $f'(x, r)$, which has a constant depth circuit, comprised of (say) NAND gates with fanin 2, and gates of a PRG, $G$. Assuming $G \in NC^1$, we could replace the $G$-gates by $NC^1$ circuits, and obtain a randomized encoding in $NC^1$. Finally, using [73],

References

[4]...
the latter encoding function can be efficiently re-encoded via a (statistically private) encoding which is a vector of degree-3 randomizing polynomials over some finite field (say, over \(F_2\)) \(^{28}\). To privately evaluate the encoding \(p(x, r)\) using any private protocol for evaluating polynomials of degree 3 (e.g. \(\Pi_{\text{priv}}\)), the parties jointly generate the randomness, and evaluate \(p(x, r_1 + \ldots + r_n)\), where \(P_i\) picks \(r_i\) independently at random. When \(G\) is only known to be in \(\text{POLY}\), this transformation does not go through, since we do not know how to perform the second step for functions outside of \(\text{NC}^1\) (additionally, as mentioned above, this approach makes non black-box use of the underlying PRG).

To summarize, the original approach is to generate a (computationally private) encoding of \(f\) via degree-3 randomizing polynomials, and then apply \(\Pi_{\text{priv}}\) to it (note that we do not rely on any structural properties of \(\Pi_{\text{priv}}\)). On a high level, to allow \(G\) be arbitrary, we modify the above construction as follows.

- Observe that Yao’s garbled circuits encoding [4] is “almost” a vector of degree-3 polynomials. More specifically, the encoding is a sequence of “plain” degree-3 polynomials \(q_i(x, r)\), “intermediate” polynomials \(q'_i(x, r)\) of the form \(q'_i(x, r) = q_i(x, r) + G(R_{i,1})\), where \(q_i\) is of degree 3, and \(R_i\) is a portion of the randomness (the formula for \(q'_i\) is a bit simplified, but it captures the flavor of the encoding). We augment this encoding to be “\(\Pi_{\text{priv}}\)-friendly”, in the sense that intermediate polynomials \(p_i\) are encrypted by secret-sharing them, and separately encrypting each share \(s_{i,j}\) as \(s_{i,j} + G(R_{i,j})\) (a similar augmented Yao encoding was first used in [41], for similar purposes).

- We augment \(\Pi_{\text{priv}}\) to privately evaluate an encoding as above. One of the issues to address in the augmented \(\Pi_{\text{priv}}\) is that of generating the randomness for the encoding. The augmented protocol is such, that each party should know some generator seeds (which are portions of the randomness), yet the protocol should remain \(t\)-private \(^{29}\).

**Garbled circuits induced encoding.** Recall the randomized encoding induced by Yao’s garbled circuit:

- Some conventions. All random variables in the encoding are random and independent unless stated otherwise. All variables denoted by lowercase letters are in \(F_2\), and all variables denoted by uppercase letters are vectors of elements over \(F_2\). All operations are done bitwise over \(F_2\). \(E_{k_1,k_2}(\cdot)\) is a symmetric encryption scheme.

- Assume wlog., that the circuit \(C\) for \(f\) is comprised only of NAND gates with fanin 2, and that all output wires exit such a gate.

- Every wire \(w\) is assigned:
  - A random mask bit \(r_w\).
  - A pair of random keys \(S_{2w}, S_{2w+1}\). Let \(y\) denote the fanout of the gate that \(w\) enters. Each \(S_x, x \in \{2w, 2w+1\}\), is comprised of \(y\) sub keys \(S_{x,1}, \ldots, S_{x,y}\), where \(S_{x,i} = S_{x,i,1}, \ldots, S_{x,i,n}\).

\(^{28}\)In particular, observe that this transformation is not black-box, since the transformation explicitly uses the circuit for \(G\), to produce the encoding via randomizing polynomials. See [4] for a proof that such a composition of encodings yields a proper (computationally private) randomized encoding of \(f\).

\(^{29}\)In particular, generating randomness in the “standard” way as above is not possible. Therefore, since there is no “clean” and generic transition from the randomized encoding to a protocol evaluating it, we will not even prove that the encoding above is private, but rather directly prove the privacy of the resulting protocol.
• For each input wire \( w \) let:
  
  - masked bit: \( m_w = r_w + b_w \), where \( b_w \) is the corresponding input bit.
  
  - key: \( S_{2w+m_w} = S_{2w} \cdot (1 - m_w) + S_{2w+1} \cdot m_w \).

  Label it by \( A_w = m_w, S_{2w+m_w} \) ("," denotes concatenation).

• Intermediate wires: Assume gate \( g \) has incoming wires \( a, b \), and outgoing wires \( g_1, \ldots, g_y \). For each \( j, l \in \{0, 1\} \) let

  - \( d_{g_i,j,l} = ((j + r_a) \text{ NAND } (l + r_b)) + r_{g_i} = 1 - (j + r_a) \cdot (l + r_b) + r_{g_i} \).
  
  - \( S_{2g_i} + d_{g_i,j,l} = S_{2g_i} \cdot (1 - d_{g_i,j,l}) + S_{2g_i+1} \cdot d_{g_i,j,l} \).
  
  - \( A_{g_i,j,l} = d_{g_i,j,l}, S_{2g_i} + d_{g_i,j,l} \).

  Outgoing wire \( g_i \) receives labels \( e_{g_i,0,0}, e_{g_i,0,1}, e_{g_i,1,0}, e_{g_i,1,1} \), where \( e_{g_i,j,l} = E_{S_{2g_i}, S_{2g_i+1}}(A_{g_i,j,l}) \). We inductively extend the definition of \( m_w \) to all wires, setting \( m_w = d_{g_1,m_w,m_w} \).

• Output wire \( w \): Label by \( e_{g_i,0,0}, \ldots \) as any intermediate wire, appending \( r_w \) as well. (For simplicity, \( A_{g_i,j,l} \) is replaced by \( d_{g_i,j,l} \), padded to the proper length).

To summarize, the various wires are labeled by evaluations of fixed degree-3 polynomials in \( x, r \), or encryptions of such polynomials (which use portions of \( r \) as the encryption key). We refer to all all variables appearing in the wire labels, except for \( e_{g_i,j,l} \)'s, which are replaced by the corresponding \( A_{g_i,j,l} \)'s as the labeling polynomials of the encoding.

**The suggested \( n \)-party protocol.** We present an augmented version of \( \Pi_{\text{priv}} \), for \( t \)-privately (with knowledge of outputs) evaluating \( f \), based on an arbitrary PRG. Intuitively, this is done by privately evaluating an encoding of \( f \) as above, for a certain instantiation of the encryption scheme \( E(\cdot) \).

• Fix a 3-multiplicative, pairwise-verifiable \((n, t)\)-scheme over \( \mathbb{F}_2 \) to be used throughout the protocol.\(^{30}\)

  We use the following encryption scheme. Let \( G \) be a PRG. We denote \( G(k) \) as \((G^0(k), G^1(k))\), where \(|G^0(k)| = |G^1(k)|\). To encrypt \( v \) under a pair of keys \( k_1, k_2 \) where \( k_b = k_{b,1}, \ldots, k_{b,n} \), we have \( E_{k_1,k_2}^{i,j}(v) = (G^1(k_{1,1}) + G^j(k_{2,1}) + s_1, \ldots, G^i(k_{1,n}) + G^j(k_{2,n}) + s_n) \), where \( s_1, \ldots, s_n \) is a sharing of \( v \) under a scheme as required by \( \Pi_{\text{priv}} \).

• Round 1: Party \( P_u \) generates and distributes among all parties (including itself) shares of the following values:

  1. Any input bit \( b_w \) it holds.
  2. For every wire \( w \), a bit \( r_{w,i} \) picked independently at random.
  3. For every \( w, i \), \( S_{2w,i,u}, S_{2w+1,i,u} \) are picked independently at random.

\(^{30}\)It can be implemented using the secret sharing scheme over \( \mathbb{F}_2 \) using the second solution, as explained in Section 3.5.3.4.
• The parties initiate $\Pi_{\text{priv}}$ for evaluating the degree-3 vector of labeling polynomials (the sharing done above corresponds to the input sharing phase of the protocol); The input variables here correspond to variables in the encoding description, except for $r_w$, which equals $\sum_{i \in [n]} r_{w,i}$.

• Round 2: The parties complete the protocol initiated in Round 1 with the following modification. Recall the message from $P_v$ for a polynomial $p$ in $\Pi_{\text{priv}}$ is parsed as $(s^p_v, M^p_v)$, where $s^p_v$ is a share of some related polynomial, and $M^p_v$ is the relevant MCDS message. In Round 2 messages concerning $A_{g_{i,j,l},P_v}$ replaces $s^p_v$ with $\tilde{s}^p_v = s_v + G^l(S_{2a+i,j,u}) + G^j(S_{2b+l,i,u})$.

• Reconstruction: Party $P_u$ evaluates the garbled circuit, proceeding in the bottom-up manner as follows:

1. Recover the evaluations of all labeling polynomials except for the $A_*$’s applying the recovery procedure of $\Pi_{\text{priv}}$. Mark the input wires as “resolved”.
2. While there exist unresolved wires, pick an unresolved wire $g_i$, such that both input wires, $a,b$, to $g$ are resolved. For each $v \in [n]$, set $s^v_{A_{a,b},m_a,m_b} = s^v_{A_{a,b},m_a,m_b} + G_{m_b}(S_{2a+m_a,i,u}) + G^{m_a}(S_{2b+m_b,i,u})$. Apply the reconstruction procedure of $\Pi_{\text{priv}}$ to the $(s^v_{A_{a,b},m_a,m_b}, M^v_{A_{a,b},m_a,m_b})$’s and the Round 2 messages. Mark the wire $w$ as “resolved”.
3. For each output wire $w$, recover an output bit $o_w = m_w + r_w = o_w + r_w + r_w = o_w$ ($r_w$ is recovered in step 1, and $m_w$ is recovered when resolving wire $w$). Output the value resulting from concatenating the $o_w$’s.

Security proof sketch. It is sufficient to prove that the protocol above is (computationally) private with knowledge of outputs assuming $G$ is a PRG. Intuitively, there are only two types of inconsistent behavior the adversary can exhibit. The first type is that one or more of the values it shares are inconsistent with a valid sharing to the uncorrupted parties. In this case it learns nothing it did not know to begin with (by privacy of $\Pi_{\text{priv}}$). Otherwise, privacy follows from the fact that $G$ is a PRG, and the proof resembles the reasoning proving the security of constructions relying on the garbled circuits technique in the literature (e.g., in [11]). More specifically, we suggest the following simulator $S_{\text{bbox}}$ for an adversary $A$.

• Let $S_{\text{priv}}$ denote the simulator for $A$ as an adversary attacking $\Pi_{\text{priv}}$ when evaluating the labeling polynomials vector, naturally extended for a vector of polynomials.

• Round 1 Simulation. Simulate the incoming Round 1 messages of the real protocol. These include random shares of the relevant variables distributed by $[n] \setminus A$, (e.g., a random sharing of 0 for all variables). Additionally, it includes Round 1 MCDS messages.

• Submitting input to the TP. Invoke $A$ on its input $x_A$, and the simulated incoming messages.

1. If the share of all variables (including random variables) to $[n] \setminus A$ are consistent with some value $u^*$, submit $x^*_A = b^*_A$.
2. Otherwise, submit $x^*_A = 0$. 

71
- **Round 2 Simulation.** Simulate the MCDS messages (the \(M^*_{[n]\setminus A}\) part) of all labeling polynomials as defined in \(S_{\text{priv}}\) (using values generated in previous steps). Let \(o\) denote the output received from the TP. The \(\tilde{s}_{[n]\setminus A}^*\) incoming messages are simulated as follows.

  - If case 2 above happened, let each \(\tilde{s}_u^o\) sent by party \(P_u\) for each polynomial be a random independent value.
  - If case 1 above happened: For each wire \(w = g_i\) (\(i = 1\) if \(w\) is an input wire):
    * Pick \(m_w\), and \(S_{2w+m_w,i,[n]\setminus A}\) independently at random.
    * The \(S_{2w+m_w,i,u}\)'s and the \(m_w\)'s determine the evaluation of
      \(A_{g_i,m_w,m_b}\) if \(w\) is intermediate, having wires \(a, b\) entering \(g_i\). If \(w\) is also an output wire, also set \(r_w = o_w + m_w\) (\(o_w\) is a bit received from the TP).
      \(A_w\) if \(w\) is an input wire.
    Denote these polynomials the “active” polynomials. Feed these evaluations to \(S_{\text{priv}}\), and simulate the \(\tilde{s}_u^o\)'s that would be sent were we running \(\Pi_{\text{priv}}\) (using values generated in previous steps) for active \(p\)'s 31.
    * Compute the \(\tilde{s}_u^o\)'s \(u \in [n] \setminus A\) for the various labeling polynomials \(p\) as follows:
      \(\text{For } p \text{ of the form } A_{g_i,m_a,m_b} (w \text{ is intermediate}), \text{set } \tilde{s}_u^p = s_u^p + G^{m_a}(S_{2a+m_a,i,u} + G^{m_b}(S_{2b+m_b,i,u})).\)
      \(\text{For active } p \text{ of other forms (only present in input and output wires), } \tilde{s}_u^p = s_u^p.\)
      \(\text{The } \tilde{s}_u^o \text{ is picked independently at random for non-active polynomials } p.\)
  
  Output the entire simulated view so far.

- **Submitting uncorrupted parties’ output:** Simulate the Round 2 messages of \(A\) (feeding it with its simulated view so far), and compute the outputs of uncorrupted parties based on \(A\)'s (incoming and outgoing) Round 2 messages (as in \(\Pi_{\text{priv}}\), a party’s output in our protocol depends only on its Round 2 messages sent by each of the parties, which are identical for all parties). Instruct the TP to send each party the computed output.

**Simulator validity sketch.** Since we get knowledge of outputs “for free” (by the protocol structure), it remains to prove that the protocol is computationally private. If case 2 above happened (\(A\) sent inconsistent shares to \([n] \setminus A\), \(S_{\text{bbox}}\) perfectly simulates \(A\)'s output distribution (by properties of \(S_{\text{priv}}\)). Otherwise, we prove that \(S_{\text{bbox}}\) computationally simulates \(A\)'s output distribution using a hybrid argument. Assume for contradiction that for some infinite input sequence \(\{x_i^{[n]\setminus A}\}_i\), there exists a polynomial \(q\), an algorithm \(D\) distinguishing between \(S_{\text{bbox}}\)'s output and \(A\)'s view in the real-world execution on input \(x_i^{[n]\setminus A}\) with advantage \(\epsilon \geq q^{-1}(1)\) for all \(x^i\). Denote them as \(D_{\text{ideal}}, D_{\text{real}}\). Let \(I = \{g_1^1, \ldots, g_{g_1}^1, \ldots, g_{g_q}^q\}\) denote the set of non-input wires. We prove that one can brake the PRG for an infinite number of input lengths (induced by the \(x^i\)'s). For some fixed \(x^i\), define the following distributions.

31Formally, we need to also feed \(S_{\text{priv}}\) with evaluations of non-active polynomials. However, since \(s_u^o\)'s for inactive \(p\)'s appear (pseudo)random and independent, and since \(s_u^o\)'s for active \(p\)'s depend only on their evaluation given by the TP (by Observation 3.6), this is sufficient.
Let $D_{0,0,0,0} = D_{\text{real}}.$

For $w = g^h_i \in I, v \in [n] \setminus \mathcal{A}, d = (d_a, d_b) \in \{(0, 1), (1, 1)\}$, with incoming wires $a, b$, labeled by $(m_a, m_b)$ define an augmentation of $\mathcal{A}$'s view as its view in the protocol, except for:

- When computing the $\tilde{s}_w^v$'s, if $d_a = 1$ replace $G(S_{2a+m_a+1,i,v})$ with a random, independent value (the same one for all $4 \mathcal{A}_{w,*,*}$'s).

- Act analogously for $G^*(S_{2b+m_b+1,i,v})$, $d_b$.

Order the tuples $(w, v, d_a, d_b)$ in numerical order (viewing it as a 4-“digit” number). Define $D_{w,v,d_a,d_b}$ to be the distribution obtained by augmenting $\mathcal{A}$’s real-world view by modifying the view according to $(w', v', 1, 1)$ as described above, for all $(w', v') < (w, v)$, and to $(w, v, d_a, d_b)$.

Observe that $D_{g^h_i,n-|\mathcal{A}|,1,1} = D_{\text{ideal}}$ (the proof is easy, and is omitted). By a standard argument the fact that $D(H_{\text{ideal}}) - D(H_{\text{real}}) \geq \epsilon$ (by the contradiction assumption) implies that for some pair of consecutive distributions $H_T, H_{\text{next}(T)}$, we have $D(H_T) - D(H_{\text{next}(T)}) \geq \epsilon/O(|C| \cdot n)$. It is not hard to fill in the details on how to take advantage of two consecutive distributions being distinguishable by the algorithm (with advantage $\epsilon/poly(|C|))$, to distinguish between $G(\text{unif})$ and the uniform distribution on strings of the proper length. Combining the conclusions for an infinite number of $x^T$’s (and thus input lengths), contradicts the fact that $G$ is a PRG. \hfill \Box
Chapter 4

Limitations on constructing perfect constant-round protocols

In this chapter we revisit the question of constant-round unconditional MPC for POLY. We show that a common general technique for devising constant-round perfectly secure protocols for POLY has limited power. On the flip side, we point out a positive implication this negative result, interpreting it as a mechanism for designing parallel algorithms.

4.1 Introduction

We study the computational power of randomizing polynomials. Randomizing polynomials represent a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ by a low-degree randomized mapping $p : \mathbb{F}^n \times \mathbb{F}^m \rightarrow \mathbb{F}^s$ over a finite field $\mathbb{F}$, such that for any input $x \in \{0, 1\}^n$ the output distribution of $p(x, r)$, induced by a uniform choice of $r$, reveals $f(x)$ and no additional information about $x$. More concretely, the latter property can be broken into privacy, requiring that the distributions $p(x, r)$ and $p(x', r)$ be identical if $f(x) = f(x')$, and correctness, requiring that the two distributions be statistically far if $f(x) \neq f(x')$. (See Section 4.1.1 below for a discussion of other variants of randomizing polynomials.)

A simple example is the function OR($x_1, \ldots, x_n$), which can be represented by a single polynomial $p(x, r) = \sum_{i=1}^n x_i r_i$ over any finite field $\mathbb{F}$. That is, perfect privacy holds since if OR($x$) = 1, $p(x, r)$ is a random field element, and is identically 0 otherwise. That is, there are two output distributions $D_0, D_1$ corresponding to outputs 0, 1 respectively. Correctness holds, as $D_0$ and $D_1$ have statistical distance $1 - \frac{1}{|\mathbb{F}|} \geq 1/2$.

Randomizing polynomials were explicitly introduced in [73] and have found various applications, mainly in the field of cryptography [73, 40, 3, 62, 63, 46]. In particular, any function with an efficient representation by constant-degree randomizing polynomials admits an efficient multiparty protocol with unconditional security and a constant number of rounds [73], and any such one-way function implies the existence of a one-way function in the complexity class NC$^0$ [3].

Motivated by the above applications, we study the class of functions $f$ admitting an efficient representa-
tion by randomizing polynomials of a constant total degree. By “efficient” we mean that the description length of the random input $r$ (and thus also $m$ and $\log|\mathbb{F}|$) is bounded by a polynomial in $n$. It is known that this class contains NC$^1$ as well as several log-space classes that are contained in NC$^2$ [73, 74, 40]. Whether this class contains all polynomial-time computable functions is a wide open question. A positive answer would have major and unexpected consequences, including the existence of efficient constant-round multiparty protocols with unconditional security, and the equivalence of (polynomial-time) cryptography and cryptography in NC$^0$.

We obtain evidence for the limited power of randomizing polynomials by showing that a useful subclass of constant-degree randomizing polynomials cannot efficiently capture functions beyond NC. Concretely, we consider randomizing polynomials over fields $\mathbb{F}$ of a small characteristic in which each monomial has degree (at most) 2 in the random inputs $r$ and degree 1 in the inputs $x$. We refer to such randomizing polynomials as being \textit{quadratic}. Most constructions of randomizing polynomials from the literature are in fact quadratic. (See Section 4.1.1 below for some exceptions.) In particular, quadratic randomizing polynomials suffice to obtain the positive results for general constant-degree randomizing polynomials mentioned above.

Our main result is that all functions $f$ which can be efficiently represented by quadratic randomizing polynomials over fields $\mathbb{F}$ of a small characteristic (say, polynomial in $n$) are in non-uniform NC. Moreover, the same holds over arbitrary fields given a Quadratic Residuosity oracle. Thus, unless Quadratic Residuosity is P-complete under NC reductions (which seems unlikely), quadratic randomizing polynomials cannot efficiently represent all polynomial-time computable functions.

Our main result is obtained in two steps: (1) we observe that computing $f$ as above reduces (via a non-uniform parallel reduction) to counting roots of a degree-2 multivariate polynomial; (2) we design parallel algorithms for the latter problem. More concretely, we obtain an NC$^2$ root-counting algorithm over fields of an odd characteristic, and an RNC$^3$ algorithm over fields of characteristic 2. These parallel root counting algorithms may be of independent interest.

On the flip side, our negative result on the power of randomizing polynomials provides an avenue for obtaining new parallel algorithms via the construction of randomizing polynomials. This gives a surprising application of cryptography to algorithm design. We include some examples for the potential usefulness of this approach, obtaining (non-uniform) alternatives to known (uniform) algorithms, but with the advantage of being based on a unified and conceptually simple approach. Our examples include parallel algorithms for Quadratic Residuosity over fields of a small characteristic, matrix rank, matrix similarity, LDU decomposition, and computing the determinant.

### 4.1.1 Related work

In this work we consider the original notion of randomizing polynomials from [73], which requires perfect privacy and statistical correctness. Several other variants of randomizing polynomials were considered in the literature, including ones that settle for statistical privacy or insist on perfect correctness. The main known facts about the complexity of randomizing polynomials are insensitive to these variations. In contrast,\footnote{The class remains the same even if the constant is restricted to 3 [73, 3].}

\footnote{The above results hold even if the statistical correctness requirement in the definition of randomizing polynomials is relaxed to only require that the output distributions $p(x, r)$ and $p(x', r)$ be distinct whenever $f(x) \neq f(x')$. The requirement of having degree 1 in $x$ can be significantly relaxed as well: it suffices that the coefficients of $p(x, r)$, viewed as a function of $r$, can be computed in NC given $x$.}
settling for computational privacy, randomizing polynomials of total degree 3 can efficiently represent all polynomial-time computable functions under standard cryptographic assumptions [4].

Our negative results on the power of randomizing polynomials apply only to quadratic randomizing polynomials \( p(x, r) \), namely ones that have degree 2 in \( r \) and degree 1 in \( x \). As noted above, most of the known positive results about randomizing polynomials can be realized by quadratic randomizing polynomials. However, there are also some natural constructions in which degree 3 in the randomness is required [74, 46]. A notable example is the boolean function (promise problem) \( f(a, N) \) determining the quadratic character of an \( n \)-bit integer \( a \) modulo an \( n \)-bit number \( N = pq \), where \( p, q \) are two primes of length \( n/2 \) bits. This function is not known to be in non-uniform NC, but it does admit an efficient representation by degree-3 randomizing polynomials over \( \mathbb{F}_2 \) [46]. This provides evidence that general degree-3 randomizing polynomials can be more powerful than quadratic ones, at least in isolated cases.

Our main technical tool is root counting of degree-2 polynomials. Root counting of multivariate polynomials is a problem of independent interest that has been previously studied. Efficient sequential algorithms for counting roots of degree-2 polynomials were obtained in [97, 47]. Also, [97] in fact implies a parallel algorithm for evaluating \( \text{NC} \) algorithms of degree 3 is already \#P-complete (over all finite fields) [47, 66]. Efficient algorithms for relaxed versions of the root counting problem, such as approximating the number of roots or counting modulo certain values, have also been considered [68, 84, 66].

### 4.1.2 Overview of techniques

As mentioned above, our core technical result is a way to devise a non-uniform NC algorithm for evaluating a boolean function \( f \) which is efficiently represented by quadratic randomizing polynomials \( q(x, r) \), namely ones that have degree 2 in \( r \) and degree 1 in \( x \). More concretely, suppose that \( f : \{0, 1\}^n \to \{0, 1\} \) and \( q(x, r) = (q_1(x, r), \ldots, q_k(x, r)) \) is over \( \mathbb{F} = \mathbb{F}_{p^\ell} \). Recall that the distribution \( q_x(r) = q(x, r) \) is some \( D_0 \) whenever \( f(x) = 0 \), and some \( D_1 \) otherwise, where \( D_0, D_1 \) are “far apart”.

As a first step, we transform the randomizing polynomials \( q(x, r) \) into randomizing polynomials over \( \mathbb{F}_p \) in the standard way, by replacing each variable \( v \) by a vector \((v_1, \ldots, v_\ell) \in \mathbb{F}_p^\ell \). Since addition in \( \mathbb{F}_p^\ell \) is coordinate-wise, and the product of \( v, u \) is a bilinear form in \( u, v \), the replacement results in a representation of polynomially related size and same degree bounds.

Our next step relies on the abelian group analogue of Vazirani’s XOR lemma from [73]. This lemma implies that for all pairs of random variables \( X, X' \in \mathbb{F}^s \), there exists a vector \( \alpha \in \mathbb{F}^s \) so that the random variables \( S = \sum_i \alpha_i X_i, S' = \sum_i \alpha_i X'_i \) have distinct distributions. Now, let \( \alpha \) be the vector guaranteed for distinguishing between \( q(x, r) \) and \( D_0 \) (here \( q(x, r) \) is already over \( \mathbb{F}_p \)). Let \( x_0 \) satisfy \( f(x_0) = 0 \). Applying \( \alpha \) results in a pair of degree-2 polynomials in \( r \), \( q_x(r) = \sum_i q_i(x, r), q_{x_0}(r) = \sum_i q_i(x_0, r) \), having distinct output distributions iff. \( f(x) = 1 \) (for uniformly picked \( r \)).

This immediately implies that \( f \) can also be “weakly represented” by a single randomizing polynomial \( q'(x, r) \) over the base field \( \mathbb{F}_p \) with the same bound on the degrees. The representation is weak in the sense that the induced distributions \( D'_0, D'_1 \) corresponding to output values 0, 1 are merely distinct, with no requirement on the distance between them.

The task of comparing the output distributions of \( q_x, q_{x_0} \), in turn, can be reduced (in parallel) to counting roots of degree-2 polynomials. For the case of small \( \mathbb{F} \), on which we focus in this overview, this can
be done by applying a root counting algorithm to \( q_r' \) \(-\) \( b, q_{x_0}' \) \(-\) \( b \) for all \( b \in \mathbb{F} \). But in fact, our root counting algorithm finds a very compact representation of the entire output distribution of the polynomial, so two distributions can be compared merely by comparing these representations. This allows the reduction to work over large fields as well. The reduction is non-uniform in the sense that generating a circuit for \( f : \{0,1\}^n \rightarrow \{0,1\} \) is not generally efficient in \( 1^n \). Thus, it remains to devise an \( NC \) algorithm for counting roots of degree-2 polynomials. This step requires the bulk of technical work in this chapter.

Remark 4.1. The main non-uniform step of the entire reduction is in moving from \( q(x,r) \) to \( q'(x,r) \) by finding \( \alpha \) (finding \( x_0, x_1 \) given \( 1^n \) could in principle be hard as well, but is typically easy for natural functions \( f \)). Our root finding algorithm is non-uniform in a weaker sense. For fields of characteristic 2, the algorithm is in fact in \( RNC^3 \), which is in turn contained in non-uniform \( NC^3 \).

For the rest of this section, we will focus on describing our root counting algorithm. We address the case of \( p = 2 \) and of odd \( p \) separately. In both cases, we first reduce root counting of general degree-2 polynomials to counting solutions of \( q(z) = b \), where \( q(z) \) is a quadratic form. A quadratic form is a degree-2 polynomial in which all monomials are of degree exactly 2. To simplify the following exposition, we only refer to the problem of counting the roots of a quadratic form \( q(x) \). The techniques for solving this problem, described below, are also useful for reducing the general case to this special case.

Let us first describe the approach from [97] for counting roots of quadratic forms, which is the starting point of our algorithms, and point out how our algorithms modify it to achieve parallelism. The root counting algorithm from [97] employs a representation of a quadratic form \( q(z) \) over \( \mathbb{F} \) as a matrix \( Q \in \mathbb{F}^{n \times n} \), such that \( q(z) = z^T Q z \). Note that the same \( q(z) \) can have many such representations.

For both odd and even characteristic, the high level idea in [97] is as follows. Observe that \( q(Cz) \) has the same number of roots as \( q(z) \) for any non-singular \( C \in \mathbb{F}^{n \times n} \), because \( C \) defines a permutation on \( \mathbb{F}^n \). We say that \( q(z), q'(z) \) are equivalent if \( q(Cz) \equiv q'(z) \) for some non-singular \( C \). A key observation is that there exists a set of “simple” quadratic forms that are canonical in the sense that every quadratic form is equivalent to some canonical form, and such that counting the roots of a canonical form is easy.

The canonical forms used in [97] admit diagonal matrix representations in the odd case, and “almost diagonal” (block-diagonal of size 2) representations in the even case. However, the process used in [97] for finding a canonical form \( Q' \) is sequential. More concretely, the computation in [97] proceeds by “revealing” \( Q’ \) one block at a time, so that in the \( i \)'th iteration a suitable (non-singular) transformation \( C_i \) is found such that \( (\Pi_{j=1}^i C_j)^T Q (\Pi_{j=1}^i C_j) \) is block-diagonal, and agrees with \( Q' \) on the first \( i \) blocks. A naive circuit implementation of this algorithm will have \( \Omega(n) \) depth. The transformation \( C \) itself is found in the process as a by-product of finding \( Q' \).

Towards describing our parallel approach, let us consider the odd and even cases separately. In the odd case, the results in [97] in fact imply a parallel root counting algorithm for any quadratic form whose unique representation by a symmetric matrix \( Q \) is non-singular. (The symmetric representation of a quadratic form \( q(z) = \sum_{i \leq j} a_{i,j} x_i x_j \) is given by \( Q_{i,j} = Q_{j,i} = 2^{-1} a_{i,j} \).) For such forms, there is a simple formula for the number of solutions to \( q(z) = b \) involving a computation of quadratic residuosity and of a determinant over \( \mathbb{F} \). This formula is computable in \( NC \) for fields of small characteristic, or given a quadratic oracle in general.

We reduce the case of a general (possibly singular) symmetric \( Q \) of rank \( t \) to the non-singular case by transforming \( Q \) into an equivalent symmetric \( Q' \) which contains a non-singular \( t \times t \) matrix in its top-left
corner and 0’s elsewhere. Since \( Q' \) represents a non-singular form in \( t \) variables, its number of roots can be counted in parallel using the formula from [97]. More concretely, the transformation of \( Q \) to \( Q' \) requires finding a submatrix \( Q_{I,I} \) of rank \( t \). This can be done by taking \( I \) to be a basis of \( Q' \)'s row space, which can be found in NC. We show that \( Q \) is equivalent to the matrix \( Q' \) consisting of \( Q_{I,I} \) as its top-left corner and 0’s elsewhere. Thus, to count the roots of \( q \), we can apply the formula from [97] to \( Q_{I,I} \) and multiply the result by \( |F|^{n-t} \). We also show how to find in NC a non-singular equivalence transformation matrix \( C \) such that \( C^T QC = Q' \). This is useful for reducing the case of general degree-2 polynomials to that of quadratic forms.

In the case of characteristic 2, the algorithm implied by [97] is only parallel for canonical forms which are block diagonal matrices with block size 2. To obtain a general parallel algorithm here, we employ a “divide and conquer” approach, which in iteration \( i \) finds an equivalence transformation \( C_i \) such that \( Q_i = C_i^T Q_{i-1} C_i \) has twice as many blocks which are twice as small. We thus obtain a canonical form equivalent to \( Q \) within \( \log n \) iterations. We implement each iteration in RNC\(^2\) by reducing it, in parallel, to solving a system of linear equations. This algorithm is more technically involved than the previous one and relies, among other things, on non-trivial properties of quadratic forms over \( F_{2^t} \), and properties of the rank distribution of alternate matrices over \( F_{2^t} \). Here as well, our algorithm finds \( C \) as a by-product, which is useful for devising a root counting algorithm for general degree-2 polynomials.

**Remark 4.2.** As we mentioned before, our starting point are the sequential algorithms for root counting for degree-2 quadratic forms over \( F \) presented in [97] (both for odd characteristic, and for characteristic 2). These algorithms output a quadratic form equivalent to a given form \( Q \), of some simple canonical form, from which deducing the root count is easy. This computation of the canonical form can be expressed as a polynomial-size, and polynomial-degree algebraic circuit with coefficients in \( F \), upto using divisions and branching. A possible alternative approach for devising a root-counting algorithm could be applying methods for division elimination to transform the circuit to one using only additions and multiplications (as done in [22] for the determinant), which is equivalent to a low degree polynomial, and applying [121]’s transformation to the resulting circuit to obtain a circuit of polylogarithmic depth. However, there is a subtlety that makes the above general approach hard to use in our case. The problem is that the canonical form computation requires to specify a concrete finite field \( F \) over which we work, rather than being the same when considering the input as given over an arbitrary extension field.\(^3\) Since the division elimination step in the generic scheme above relies on the fact that the computation is the same for all extension fields, it seems unsuitable in our case, and an ad-hoc approach is required.

**Organization** Section 2.5 presents some notation we will need. Section 4.2 contains some additional technical background, which is specific to this chapter. Section 4.4 presents our parallel root-counting algorithm over fields of an odd characteristic. The case of characteristic 2, which is more technically involved, is deferred to Section 4.7. Section 4.5 establishes the relation between root counting and randomizing polynomials, and in Section 4.6 we present several examples of applying randomizing polynomials towards the design of parallel algorithms.

\(^3\)This is as opposed to the Gaussian elimination-based algorithm for the determinant, which is oblivious to the concrete extension field we work over.
4.2 Preliminaries

4.2.1 Polynomials

We consider multivariate polynomials over finite fields. By “degree-\(d\) polynomial” we refer to a polynomial whose total degree is at most \(d\), namely one that can be written as a sum of monomials such that each monomial is a product of at most \(d\) (not necessarily distinct) variables.

We define the signature of a polynomial \(q(x_1, \ldots, x_n)\) over \(\mathbb{F}\) as the function \(\operatorname{sig}_q(b) : \mathbb{F} \to [0, 1] \) mapping each \(b \in \mathbb{F}\) to the fraction of inputs \(x \in \mathbb{F}^n\) for which \(q(x) = b\). We let \(\#_q(b)\) denote the number of solutions \(x\) to the equation \(q(x) = b\). Note that \(\#_q(b) = |\mathbb{F}|^n \cdot \operatorname{sig}_q(b)\).

Two polynomials \(q_1(x), q_2(x)\) are said to be equivalent if there exists a non-singular matrix \(C\), such that \(q_1(x) \equiv q_2(Cx)\) (that is, the two polynomials are identical). We refer to such a \(C\) as an equivalence transformation. Note that equivalent \(q_1, q_2\) have the same signature, since the mapping \(x \mapsto Cx\) is a permutation over \(\mathbb{F}^n\). It is easy to see that this is indeed an equivalence relation.

A degree-2 polynomial \(q(x_1, \ldots, x_n)\) is called a quadratic form if the degree of each monomial is exactly 2. Equivalently, \(q(x_1, \ldots, x_n)\) can be represented as \(q(x) = x^T Q x\), where \(Q \in \mathbb{F}^{n \times n}\) satisfies \(Q_{i,j} + Q_{j,i} = q_{i,j}\) for \(i < j\), and \(Q_{i,i} = q_{i,i}\). We will use the polynomial-based and the matrix-based notation interchangeably. A quadratic form \(A \in \mathbb{F}^{n \times n}\) is called alternating if \(x^T Ax \equiv 0\) (equivalently, \(A\) is antisymmetric and its diagonal is 0).

**Observation 4.1.** If \(Q_1 = C^T Q_2 C + A\) for \(A, C, Q_1, Q_2 \in \mathbb{F}^{n \times n}\), where \(C\) is nonsingular and \(A\) alternating, then \(Q_1, Q_2\) are equivalent.

**Proof.** By definition \(x^T Ax \equiv 0\). Therefore, \(q_1(x) = x^T C^T Q_2 C x + x^T A x = x^T C^T Q_2 C x = q_2(Cx)\). □

A quadratic form \(Q\) is regular if \(Q\) is not equivalent to any \(Q'\) which depends on less than \(n\) variables (a polynomial \(q\) does not depend on a subset \(I \subseteq [n]\) of its variables, if the value of \(x_I\) determines \(q(x)\)). Alternatively, there is no equivalent \(Q'\), where row \(n\) and column \(n\) are all 0.

4.2.2 Parallel algorithms

We consider families of functions of the form \(f : \mathbb{F}^n \to S\), where \(\mathbb{F}\) is a finite field, and \(S\) is some output domain. While the field size may grow with \(n\), its characteristic will usually remain fixed. We say that the family is in \(\text{NC}^i\) if it can be computed by arithmetic circuits of size \(\text{poly}(n)\) and depth \(O(\log^i n)\) over the field \(\mathbb{F}\). Such a circuit can contain bounded fan-in gates of arithmetic operations (addition, multiplication and inverse) and equality tests over \(\mathbb{F}\), where the inputs to the gates are either variables or constants from \(\mathbb{F}\). When the field size is fixed, this class corresponds to the boolean complexity class \(\text{NC}^i\). The class \(\text{NC}\) is defined as the union of all \(\text{NC}^i\). We use \(\text{RNC}^i\) to denote the randomized version of \(\text{NC}^i\). Unless otherwise noted, we assume circuit families to be polynomial-time uniform.

A function family \(f\) is \(\text{NC}^i\)-reducible to a function family \(g\) if there exist \(\text{NC}^i\) circuits for \(f\), which can be augmented with (unbounded fan-in) gates for evaluating \(g\), such that every path from an input wire to an output wire includes only a constant number of \(g\)-gates. If \(g \in \text{NC}^j\) and \(f\) is \(\text{NC}^i\)-reducible to \(g\), then \(f \in \text{NC}^{\max(i,j)}\).
We rely on some standard parallel algorithms from the literature, and include details on some of the less standard variants for self-containment.

1. Computing the rank of a matrix $M \in \mathbb{F}^{m \times n}$, the determinant of a matrix $M \in \mathbb{F}^{n \times n}$, and (a basis for) the solution space of a linear system $Ax = b$ are all in NC$^2$ [104, 22].

2. Given $x \in \mathbb{F}_p^n$, where $p$ is a fixed prime, evaluating $x$ to any power $y \in [p^n]$ is in NC$^2$ [51]. This implies an NC$^2$ algorithm for deciding quadratic residuosity in fields of a fixed odd characteristic (given $x$, compute $x^{(p^n-1)/2}$). In contrast, deciding quadratic residuosity in $\mathbb{F}_p$, where $p$ is an $n$-bit prime, is neither known to be in NC nor P-complete under NC-reductions.

3. Given a set of vectors $R = \{v_1, \ldots, v_t\} \in \mathbb{F}^n$, a basis for the space spanned by $R$ can be found in NC$^2$ [22]. This is done via the following reduction to rank: Let $M = [v_1 \ldots | v_2]$. Compute the rank of the $t$ submatrices $M_{[n],[i]}$ in parallel. Pick the vectors $v_i$ for which $\text{rank}(M_{[n],[i]}) > \text{rank}(M_{[n],[i-1]})$.

4. Given two subspaces $U \subseteq V \subseteq \mathbb{F}^n$, specified by bases $\{v_1, \ldots, v_t\}$ and $\{u_1, \ldots, u_{t'}\}$ respectively, complementing $U$ into a basis of $V$ can be done by the following NC$^2$ algorithm. Let $M_V = [u_1 \ldots | u_{t'} | v_1 \ldots | v_t]$, and find a basis for the column space of $M_V$ using the procedure from the previous item.

### 4.3 Some required algebra

This section contains some mathematical theory used throughout this chapter. Some of it is standard, and some of it is new (as the analysis of the rank distribution of alternate matrices).

#### 4.3.1 Linearized polynomials

Polynomials in $\mathbb{F}_2[x]$ of the form $p(x) = \sum_{i=0}^{t-1} a_i x^{2^i}$, called “linearized polynomials”, are a linear mapping from $\mathbb{F}_{2^t}$ to $\mathbb{F}_2$ (see [97] for more details on linearized polynomials). Observe that finding roots of such polynomials is in NC$^1$, using the following algorithm:

- Find a matrix $M \in \mathbb{F}_2^{t \times t}$ such that $Mx = p(x)$ to this end, evaluate $p$ at $x_1, \ldots, x_t$ represented by $e_1, \ldots, e_t$ (as vectors over $\mathbb{F}_2$). Thus $M = [p(x_1) | \ldots | p(x_t)]$. The evaluations can be made efficiently for arbitrary $p$, but since we will need only degree-2 $p(x)$’s, it is sufficient to consider the case of constant degree $p(x)$. The latter follows since arithmetic (additions and multiplications) over $\mathbb{F}_{2^t}$ are in NC$^1$.

- Output a basis of $M$’s right kernel.

#### 4.3.2 Properties of quadratic forms

**Observation 4.2.** Let $Q \in \mathbb{F}^{n \times n}$ be a quadratic form.

1. Every quadratic space $(V, q)$ where $V \subseteq \mathbb{F}^n$, $V = \text{colSpan}(M_V)$, and $M_V \in \mathbb{F}^{n \times t}$ is of rank $t$, is equivalent to $(\mathbb{F}^t, q')$, where $Q' = M_V^T Q M_V$. 

80
2. \( q(x + v) = q(x) \) for all \( x \in \mathbb{F}^n, v \in S_Q \).

Proof. Part 1 follows by considering the linear mapping \( c \), induced by setting \( c : V \to \mathbb{F}^l \), where \( c(M^{(i)}_V) = e_i \). For part 2, we have

\[
q(x + v) = (x + v)^T Q(x + v) = q(x) + q(v) + x^T Q v + v^T Q x = q(x) + q(v) + x^T (Q + Q^T) v = q(x).
\]

Here the last equality follows from \( v \in S_Q \).

We “single out” the following classes of quadratic forms.

Definition 4.1.

- Type 0: \( q'(x_1, \ldots, x_n) = \Sigma_{i \leq k} (x_{2i} - 1)x_{2i} + a_i (x_{2i-1}^2 + b_i x_{2i}^2) \) for some \( 2k \leq n \), where \( \text{Tr}_{2^\ell}(b_i) = 1 \), and \( a_i \in \{0, 1\} \). We define \( \text{Arf}(q') = \Sigma_i a_i \).
  - If \( \text{Arf}(q') = 0 \), we say \( q' \) is of type 0.0.
  - Otherwise, we say it is of type 0.1.

- Type 1: \( q'(x_1, \ldots, x_n) = \Sigma_{i \leq k} x_{2i-1}x_{2i} + x_{2k+1}^2 \) for some \( 2k < n \).

In the following lemma, we prove that the forms in Definition 4.1 are canonical in the sense that each quadratic form is equivalent to a form as in Definition 4.1 (in the sequel, they will be referred as “canonical”). Furthermore, we show that solution counting for canonical forms is “easy” (by providing an expression for computing it).

Lemma 4.1.

1. A quadratic form \( q \in \mathbb{F}_2^{n \times n} \) is always equivalent to a canonical form \( q' \) as in definition 4.1.

2. For canonical forms:
   - Type-0 forms \( q' \) (both 0.0 and 0.1) satisfy
     \[
     \text{sig}_{q'}(0) = 2^{-\ell} + (2^{\ell} - 1)(1 - 2\text{Arf}(Q')) \cdot 2^{-\ell(1+k)},
     \]
     \[
     \text{sig}_{q'}(b) = 2^{-\ell} - (1 - 2\text{Arf}(Q')) \cdot 2^{-\ell(1+k)} \quad \text{for } b \neq 0.
     \]
   - Type-1 forms \( q'(x_1, \ldots, x_n) \) for some \( 2k < n \), satisfy \( \text{sig}_{q}(b) = 2^{-\ell} \) for all \( b \in \mathbb{F}_{2^\ell} \).

It follows from the lemma that \((k, T)\), where \( T \in \{0.0, 0.1, 1\} \) is a type as in Definition 4.1 are invariants of \( Q \) in the sense that all canonical forms to which \( Q \) is equivalent have the same \((k, T)\). This follows from the fact that different \((k, T)\) pairs have different signatures. Thus, we extend the definition of \((k, T)\) for arbitrary forms \( Q \), to equal the parameters \((k, T)\) of a canonical form equivalent to it.
Our proof relies on a proof of a similar Lemma in [97]. They define canonical forms, and (implicitly) classify them into types. More specifically, their canonical forms are defined like ours, and are classified into types in the same way, with the sole extra restriction that $a_i = 0$ for all $i < k$ in type 0. We refer to the corresponding forms as LN-canonical forms, and to their parameters $(k, T)$ as LN-parameters. They prove the following variant of Lemma 4.1, which will be useful to prove our variant.

**Lemma 4.2.**

1. Every regular quadratic form $q$ is equivalent to an LN-canonical form $q'$ with $k = \lfloor n/2 \rfloor$ [97, Theorem. 6.30].

2. The signatures of the regular LN-canonical forms are as implied by Lemma 4.1 for their type by Lemma 4.1 [97, Theorem. 6.32].

**Proof.** As to part 1, for regular $q$, the existence of $q'$ as in part 1 of the lemma follows directly from Lemma 4.2, part 1. As to general $q$, by definition of “regular” there exists a $Q'$ equivalent to $Q$, where all entries in $Q'[\lfloor n \rfloor \times \lfloor n \rfloor]$ are 0, and $Q'' = Q'[\lfloor n \rfloor \times \lfloor n \rfloor]$ is regular. The result now follows by observing $Q''$ has an equivalent canonical form (and using transitivity of quadratic form equivalence).

For Part 2 of Lemma 4.1, we need the following claim.

**Claim 4.1.** The signature of a canonical (by Definition 4.1) form $q$ equals that of an LN-canonical form $q'$ with the same $(k, T)$ parameters.

Part 2 now follows directly from Lemma 4.2 and Claim 4.1. To prove Claim 4.1, it is sufficient to observe that $Q$ with parameters $(k, T)$ by our definition, also has LN-parameters $(k, T)$. Let $q_1(x_1, x_2)$ denote a polynomial of the form $x_1 x_2 + x_1^2 + x_2^2$ where $Tr_{q_1}(a) = 1$ (the concrete value of $a$ is not important), and let $q_2(x_1, x_2) = x_1 x_2, q_{22}(x) = q_2(x_1, x_2) + q_2(x_3, x_4)$. We rely on the following technical observation:

**Claim 4.2.** The signatures $sig_{q_11}, sig_{q_22}$ for $q_{11}(x) = q_1(x_1, x_2) + q_1(x_3, x_4)$ and $q_{22}(x) = x_1 x_2 + x_3 x_4$ are identical.

The result for type 0 then follows, since:

- For $Arf(q) = 0$, $q$ is the sum of $t \geq 0$ (non-overlapping) portions of the form
  \[ q_{11}(x_{2i-1}, x_{2i}, x_{2j-1}, x_{2j}) , \]
  and $k - 2t$ “pairs” of the form $q_2(x_{2i-1}, x_{2i})$. Consider an LN-canonical form $q'$, which a sum of $k$ portions of the form $q_2(x_{2i-1}, x_{2i})$, which can be organized into $t$ portions of the form $q_{22}(x_{2i-1}, x_{2i}, x_{2j-1}, x_{2j})$, and $k - 2t$ “pairs” of the form $q_2(x_{2i-1}, x_{2i})$. Matching the former with the former, and the latter with the latter, and using Claim 4.2 and the fact that the variables in different portions are disjoint, the result follows.

- For $Arf(q) = 1$, match a $q_1(x_{2i-1}, x_{2i})$-pair in $q$ with the $q_1(x_{2k-1}, x_{2k})$ pair in $q'$. The remaining parts of $q, q'$ now correspond to the previous case, and similar arguments prove that $q, q'$ have the same signatures.
Finally, Claim 4.2 follows by a simple calculation. Making the calculation explicit, the signatures are distributed like the random variables $Y_1 = y_1,1 + y_1,2$ versus $Y_2 = y_2,1 + y_2,2$, where the $y_2,i$'s are independently distributed according to $q_1$, and the $y_1,i$'s according to $q_2$. Denote $\epsilon = (2^\ell - 1)2^{-(1+k)}$. We have $P(Y_1 = 0) = (2^\ell - \epsilon)^2 + (2^\ell - 1)(2^\ell + \epsilon/(2^\ell - 1))^2$, and $P(Y_2 = 0) = (2^\ell + \epsilon)^2 + (2^\ell - 1)(2^\ell - \epsilon/(2^\ell - 1))^2$ (using Lemma 4.2 applied to $q_1$), which are equal. The computation of $P(Y_1 = y), P(Y_2 = y)$ for $y \neq 0$ is similar.

For general canonical $Q$, it is equivalent to a canonical $Q'$, where $Q'' = Q'[t]_i [t]$ is regular and canonical, and $Q'$ is 0 elsewhere for $t = 2k+T$. Thus, $q$ and $q''$ have the same signature, and in particular $Q'$'s signature is as stated in the Lemma.

**Corollary 4.1.** For a quadratic form $Q \in \mathbb{F}^{n \times n}$, we have that $K_Q$ is a (possibly empty) subspace of dimension $n - 2k$. Also, $S_Q$ is a subspace of $K_Q$ of dimension $\max(0, n - 2k - T)$, where $T \in \{0, 1\}$ is the type number of $Q$.

**Proof.** We observe that if $Q', Q$ are equivalent via $C$, then $C$ maps $K_{Q'}, S_{Q'}$ onto $K_Q, S_Q$ and vice versa (via $C^{-1}$). Thus, it is sufficient to prove the claim for canonical $Q' \in \mathbb{F}^{n \times n}$.

Clearly, if non-empty, $K_{Q'} = \text{Span}\{e_{2k+1}, \ldots, e_n\}$, and $S_{Q'} = \text{Span}\{e_{2k+T+1}, \ldots, e_n\}$, and the result follows.

### 4.3.3 On distributions of alternate matrices

We will also need a quantitative result on the rank distribution of alternate matrices. Denote by $\text{Alt}_{\mathbb{F}, n}$ the set of alternate $n \times n$ matrices over $\mathbb{F}$ ($\mathbb{F}$, $n$ will be omitted when clear from the context). We refer to $D(A) = n - \text{rank}(A)$ as $A$'s rank deficiency, and let $\text{Alt}_{\mathbb{F}, n}^d = \{h \in \text{Alt}_{\mathbb{F}, n}, D(h) = d\}$.

**Markov chains** A Markov chain $M$ is specified by a (possibly infinite) countable set $S = \{s_1, s_2, \ldots\}$ of states (vertices), and a directed graph $G_M(S, E)$, such that each $s_i \in S$ has a finite number of outgoing edges $\{(s_i, s_j)\}_j$ in $E$, where each $(s_i, s_j)$ is labeled by $m_{i,j} > 0$, and $\Sigma_{(s_i, s_j) \in E} m_{i,j} = 1$. A state of $M$ is a probability distribution $p$ over $S$. By applying $M$ to a state $p$, we mean setting $p_i = \Sigma_{(s_j, s_i) \in E} P_j \cdot m_{j,i}$, we denote this operation by $M(p)$. We say that a Markov chain $M$ is irreducible if $G_M$ is strongly connected. Finally, we say that $p$ is a stable state of $M$, if $M(p) = p$.

**Theorem 4.2.** [109] If an irreducible Markov chain has a stable state $p$, then for any initial state $q$, we have

$$\lim_{n \to \infty} M^n(q)_i = p_i$$

for all $s_i \in S$ (thus, $p$ is unique).

**Rank distributions** In this section, for a given matrix size $n$ ($n, \mathbb{F}$ will be omitted whenever clear from the context), let

$$A_{2i} = \text{blocks}(\text{blocks}(\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, i), 0_{n-2i})$$

where $0_{n-2i}$ is the all-0 matrix of size $n-2i$, and $\text{blocks}(M, i)$ denotes the block-diagonal matrix comprised of repeating $M$ $i$ times.
Lemma 4.3. Let $A$ be random in $Al_{\mathbb{F}_{2^t},n}$, where $n$ is even. Then $D(A)$ is even (with probability 1). There exists a constant $N_\ell$ such that. For $\ell = 1$, $\Pr[D(A) = 2] \geq 0.545, \Pr[D(A) = 0] \geq 0.408$ for $n \geq N_\ell$. For $\ell > 1$, we have $\Pr[D(A) = 0] > 0.735$ for $n \geq N_\ell$.

Proof.

In the course of the proof, we infer the distribution of $D(A)$ when adding a random row and column to an alternate $A$ to obtain an alternate $A'$. We then use it to build a Markov chain with even $D(A)$-values as states, having two connected components - one for the odd values, and one for the even values. The important point here is that the chain describes the new distribution after such an $A \Rightarrow A'$ transformation, based only on $D(A)$, regardless of the size of $A$. It will hold for the “even” component that its stable state $p$ satisfies that $\lim_{n \to \infty}|p_{2i} - \Pr[A \in Al_{\mathbb{F}_{2^t},n}^2]| = 0$ for $0 \leq 2i \leq n$ (a similar situation occurs for the “odd” component, but it is of less interest to us). Finally, we analyze the chain’s stable state, to obtain our result, for large enough $n$.

For $n = 2$, $\Pr[D(A) = 2] = |\mathbb{F}|^{-1}$ (the all-0 matrix), and $\Pr[D(A) = 0] = 1 - |\mathbb{F}|^{-1}$ (otherwise). Now, consider a random $A \in Al_{\mathbb{F}_{2^t},n}$ with $d = D(A)$. We obtain a matrix $A' \in \mathbb{F}^{n \times n}$ at random, and setting $A'_{n+1,n} = (A; a^T)$, and then $A'_{[n+1],n+1} = (a, 0)$. Clearly, $A'$ is random in $Al_{\mathbb{F}_{2^t},n+1}$. There are two cases. In the first case, $a^T = \alpha^T A$ (that is, $a$ is in the row span of $A$). Observe that process of generating $A'$ in this case is equivalent to setting $A'_{[n+1],n} = (A; \alpha^T A)$, $A'_{[n+1],n+1} = A'_{[n+1],n+1}$ since indeed $A'_{[n+1],n+1} = \alpha^T A \alpha = 0$, because $A$ is alternate. Also, clearly, the rank does not increase in both the first and the second transition, and $D(A') = D(A) + 1$. This case occurs with probability $|\mathbb{F}|^{-D(A)}$. In the second case $a$ is not in $row(Span(A))$, so the rank clearly increases by 1 in each of the transitions above, and $D(A') = D(A) - 1$. This case happens with probability $1 - |\mathbb{F}|^{-D(A)}$. This proves the claim on the parity of $D(A)$ by induction.

Next, we define an (infinite) Markov chain, with state set $S = \mathbb{N}$ defined by

$$
\text{Pr}[D(A) = 0] = \text{Pr}[D(A) = 1](1 - |\mathbb{F}|)$$

$$
\text{Pr}[D(A) = i] = \text{Pr}[D(A) = i - 1](1 - |\mathbb{F}|^{-i}) + \text{Pr}[D(A) = i + 1]|\mathbb{F}|^{-i-1} \text{ for } i \neq 0.
$$

We start with the initial state $p(0) = 1/|\mathbb{F}|, p(2) = 1 - 1/|\mathbb{F}|$, and make $n - 2$ iterations to obtain the required distribution for $n$. As we are only interested in even $n$, the recurrence is described by an (infinite) chain $M^2$ obtained from making pairs of consecutive steps. More concretely, we start from the distribution $\Pr[D(A) = 2] = 1/|\mathbb{F}|, \Pr[D(A) = 0] = 1 - 1/|\mathbb{F}|$ (and 0 otherwise), describing the distribution of $D(A)$ of a random matrix in $Al_{\mathbb{F},2}$. Iterating the chain $M^2 n/2 - 1$ times, we obtain exactly the rank distribution of $Al_{\mathbb{F},n}$. By the structure of $M$, $M^2$ consists of two irreducible chains - the one containing the even states, and its complement. As our starting vector is distributed over the even states, we consider only the even component, and refer to it as $M^2$ (with state set $S' = 2\mathbb{N}$).

Claim 4.3 The (infinite) chain $M^2$ has a stationary distribution $p$ that satisfies $p_0 \geq 0.409, p_0 + p_2 \geq 0.956$

---

\footnote{This is a restatement of a well known fact, but here we get it “for free”, based on the arguments above, needed to prove the rest of the lemma.}
for \(|F| = 2\), and \(p_0 \geq 0.736\) for \(|F| = 2^l > 2.\)

**Proof.** A stable state \(p\) satisfies the following equations:

\[
\begin{align*}
p_0 &= (1 - 2^{-\ell})p_0 + (1 - 2^{-2\ell})(1 - 2^{-\ell})p_2 \\
p_{i+2} &= 2^{(2i-1)\ell}p_i + (2^{-i\ell}(2^{-2\ell} + 2^{-\ell}) + \\
&\quad 2^{-2i\ell}(2^{-5\ell} + 2^{-3\ell}))p_{i+2} + (1 - 2^{-i\ell}(2^{-4\ell} + 2^{-3\ell}) + 2^{-(2i+7)\ell})p_{i+4}
\end{align*}
\]

for \(i = 0, 2, 4, \ldots\)

It is easy to prove by induction, that any solution to this (infinite) set of equations satisfies \(p_i/p_{i+2} = (1 - |F|^{-i-1} - |F|^{-i-2} + |F|^{-2i-3})|F|^{i+2i}\) for \(i = 0, 2, \ldots\). More specifically, we prove the formula for \(p_0/p_2\) as a base case, and prove it for \(i \geq 2\) assuming it holds for \(i - 2\) by simple calculation. Clearly, this ratio is monotonously increasing, and satisfies \(p_2/p_4 \geq 24.25\) for all \(F\), so setting \(p_0\) to some positive value results in a convergent series \(p_0, p_2, p_4, \ldots\) (this would hold even if the ratio from some point was some constant \(> 1\), rather than increasing). Thus, picking the right \(p_0\) determines a stationary distribution. For \(F_2\) we have \(p_0 = 3/4p_2\) and \(p_2 = 24.25p_4\). We conclude that \(\Sigma_{j \geq 2} p_{2j} \leq 1/23.25p_2 \leq 1/23.25\). Thus \(p_0 + p_2 \geq 22.25/23.25 \geq 0.956\). In particular, \(p_0 \geq 0.956 \cdot 3/7 \geq 0.409\). For \(|F| \geq 4\), we have \(p_0 = 45/16p_2\), with the ratio \(p_i/p_{i+2}\) monotonously increasing. Similarly to the above, the series \(p_0, p_2, p_4, \ldots\) converges due to \(\Sigma_{j \geq 2} p_{2j} \leq p_2/1003 \leq 1/1003\) (since \(p_2/p_4 > 1004\)). Thus, \(p_0 \geq 1002/1003 \cdot 45/61 \geq 0.736\).

Combining with Theorem 4.2, we conclude that our initial state converges to a stable state \(p\) as in the claim. Getting back to distribution of \(D(A)\) over \(A_{F_2,n}\) - it is obtained by iterating \(M^{l^2}n/2 - 1\) times from a certain starting state. The fact that the chain converges to a stable state implies that there exists some \(N\), such that after \(N\) or more iterations, \(\Pr[D(A) = i] - p_i \leq 0.001\) for all \(i\). We conclude that the statement of Lemma 4.3 holds for \(N = 2N_l + 2\).

**Lemma 4.4.** Let \(B\) be a distribution as in Theorem 4.3, and \(H_{2i} = \{C | \text{rank}(C^T A_n C) = 2i\}\). Then picking \(y\) from \(D\), the probability of \(y = D\) for a fixed alternate \(D\) of rank \(i \leq n\) is the same probability for all such \(D\), and equals 0 for odd ranks. For \(2i = n - 2\)

\[\{C | \text{rank}(C) = n - 1\} \subseteq H_{2i}\]

For \(2i = n, H_{2i} = \{C | \text{rank}(C) = n\}\).

**Proof.** The claim about odd ranks follows from Lemma 4.3, since \(C^T A C\) is alternate for all \(C\).

**Claim 4.4.** Let \(D \in F_{2^l}^{n \times n}\) be a rank-\(2i\) matrix. Then there exists a non-singular \(C \in F_{2^l}^{n \times n}\) such that \(C^T DC = A_{2i}\).

To prove the claim, denote \(D = X + X^T\), for some upper-triangular \(X\). By Lemma 4.1, there exists a similarity transformation \(C\) such that \(C^T X C\) is a canonical form. It must be the case that \((C^T X C)^T + (C^T X T) = C^T (X^T + X) C = C^T D C = A_{2i}\), or else \(\text{rank}(C^T D C) \neq \text{rank}(D)\), which cannot hold, since \(C\) is non-singular.

---

\(^5\)Curiously enough, our analysis reveals that 0.409 is the limit of the non-singular alternate matrices fraction over \(F_2\) as \(n\) grows, compared with 0.288 for general matrices.
Denote \( H_D = \{ C \mid D = C^T A_n C \} \) for a rank-2i \( D \). We prove that \( |H_D| \) depends only of \( i \). For this purpose, consider alternate \( D_1, D_2 \) of rank 2i. By claim 4.4, we have \( C_1 D_1 C_1 = A_{2i}, C_2 D_2 C_2 = A_{2i} \) for some non-singular \( C_i \)’s. Thus, \( D_1 = C^{-1}_1 C_2^T D_2 C_2 C_1^{-1} \). Therefor, for all \( C \in H_{D_i} \) we have \((C C_1 C_2^{-1})^T A(C C_1 C_2^{-1}) = D_2 \). Thus, \( h_{1,2}(C) = C C_1 C_2^{-1} \) is a 1-1 mapping from \( H_{D_1} \) into \( H_{D_2} \) (as \( C_1, C_2 \) are non-singular). Since a similar mapping exists from \( H_{D_2} \) to \( H_{D_1} \), these sets have the same size. As \( D_1, D_2 \) are arbitrary rank-2i matrices, we are done. Now, clearly, for 2i = n, \( H_{2i} = \{ C \mid \text{rank}(C) = n \} \) by properties of rank. To show that \( \{ C \mid \text{rank}(C) = n - 1 \} \subseteq H_{2i} \), observe that \( \text{rank}(C^T A_n C) \leq \text{rank}(C) = n - 1 \). However, since \( \text{rank}(C^T A_n C) \) is alternate its rank is even, so \( \text{rank}(C^T A_n C) \leq n - 2 \). On the other hand, it is a standard fact from linear algebra, that

\[
\text{dim}(\text{Ker}(C^T A_n C)) \leq \text{dim}(\text{Ker}(C^T A_n)) + \text{dim}(\text{Ker}(C)) = 2.
\]

Applying the rank-nullity theorem to the linear mapping \( C^T A_n C \), we have \( \text{rank}(C^T A_n C) \geq n - 2 \). Combining the two inequalities, it follows that \( \text{rank}(C^T A_n C) = n - 2 \), implying

\[
\{ C \mid \text{rank}(C) = n - 1 \} \subseteq H_{2i}.
\]

The main theorem our construction is based on is:

**Theorem 4.3.** Let \( n = 2k \geq N, \ell \geq 1, B = C^T A_n C \) a distribution where \( C \) is a random matrix in \( \mathbb{F}_2^{n \times n} \), \( N \) is as in Lemma 4.4. Then if \( x_1, x_2 \) are sampled independently at random from \( B \),

\[
\Pr[\text{rank}(x_1) = n, \text{rank}(x_1 + x_2) = n] \geq 0.028.6
\]

**Proof.** We start with the case of \( \ell = 1 \), and \( n \geq N_1 \), where \( N_1 \) is as in Lemma 4.3). By Lemma 4.3, the set of non-singular alternate matrices constitutes a \( \geq 0.4 \) fraction of \( A_{2, n} \), and the set of matrices with rank deficiency \( > 2 \) constitutes a \( \leq 0.05 \) fraction of \( A_{2, n} \). Fix some \( x_1 \in A_{2, n}^0 \). Then, \( T(h) = x_1 + h \) is a permutation on \( A_{2, n} \). Thus, the multiset \( \text{Good} = T^{-1}(A_1^0) \) contains each element once, and must consist of at least a \( 0.4 - 0.05 = 0.35 \) fraction that falls in \( A_1^0 \cup A_1^2 \) (rank deficiency \( \leq 2 \)). On the other hand, from Lemma 4.4 we conclude that at least a \( 0.35 \cdot 0.288 \geq 0.1 \) fraction of \( x_2 \)'s satisfies \( T(x_2) \in A_1^0 \). This is so by the fact the the distribution \( B \) restricted to \( A_1^2 \) (\( A_1^0 \)) is uniform with probability at least

\[
\text{prob}_{C \in \mathbb{F}_2^{n \times n}}(\text{rank}(C)) = n - 1
\]

(\( \text{prob}_{C \in \mathbb{F}_2^{n \times n}}(\text{rank}(C)) = n \)), and 0.288 is an easily derived lower bound on both, using the commonly known 0.288 lower bound on the fraction of non-singular matrices in \( \mathbb{F}_2^{n \times n} \). As this holds for all choices of a non-singular \( x_1 \), we conclude that a

\[
\Pr[x_1 \in A_1^0] \Pr[x_1 + x_2 \in A_1^0 \mid x_1 \in A_1^0] = 0.288 \cdot 0.1 \geq 0.028
\]

\(^6\)A better bound can be achieved for all larger fields, and it approaches 1 as \( \ell \) grows.
fraction of the \((x_1, x_2)\)'s satisfies the conditions of Theorem 4.3, and we are done. The proof for \(\ell > 1\) is even simpler. By Lemma 4.3, we have \(|A^0|/|A| \geq 0.735\), so fixing \(x_1 \in A^0\), \textit{Good} (defined by \(T, x_1\)) satisfies \(|\textit{Good} \cap A^0| \geq |A^0| - |A| \geq 0.47|A|\). On the other hand, the probability of a random \(C \in \mathbb{F}_q^{n \times n}\) being full-rank is \(\prod_{i=1}^n (1 - \frac{1}{2^{2r}}) \geq 1 - \sum_{i=1}^{\infty} \frac{2^{-i\ell}}{2^{2r-1}} \geq 2/3\) (where the worst case is obtained for \(\ell = 2, \ell = 3\) already has an estimate of \(7/8\), e.t.c). By reasoning similar to the above, \(x_2 \in \textit{Good}\) with probability

\[
\Pr[x_1 \in A^0] \Pr[x_1 + x_2 \in A^0 | x_1 \in A^0] \geq 2/3 \cdot 0.47 \geq 0.31,
\]

which satisfies the conditions of Theorem 4.3, and we are done.

\[\square\]

### 4.4 Root counting for odd characteristic

In the following sections we will consider the following root counting problem:

- **Input**: A degree-2 polynomial \(q(x_1, \ldots, x_n)\) over \(\mathbb{F}\).
- **Output**: The number of \(x \in \mathbb{F}^n\) such that \(q(x) = 0\).

We will prove the following main theorem.

**Theorem 4.4.** For any fixed prime \(p\), the root counting problem for degree-2 polynomials over fields \(\mathbb{F}\) of characteristic \(p\) is in \(\text{NC}^2\) if \(p\) is odd or in \(\text{RNC}^3\) if \(p = 2\). If \(p\) is odd and can grow with \(n\), the problem is in \(\text{NC}^2\) given an oracle computing quadratic residuosity modulo \(p\).

We treat the cases of odd characteristic and of characteristic 2 separately. The resulting algorithms and techniques turn out to be quite different, although the high-level approach is similar (with the case of odd characteristic being technically simpler). In this section we address the case of odd characteristic, and defer the case of characteristic 2 to Section 4.7.

#### 4.4.1 Quadratic forms

We start by addressing the special case of quadratic forms. In this section, we represent a quadratic form \(q(x) = \sum_{i \leq j} q_{i,j} \cdot x_i x_j\) by a symmetric matrix \(Q\) such that \(q(x) \equiv x^T Q x\). This is possible by letting \(Q_{i,j} = Q_{j,i} = 2^{-1} \cdot q_{i,j}\) and \(Q_{i,i} = q_{i,i}\). Note that such a symmetric representation may not exist over fields of characteristic 2.

We say that a quadratic form \(Q\) is \textit{canonical}, if it is block diagonal with \(Q = \text{blocks}(Q_{[t],[t]}; 0)\), where \(Q_{[t],[t]}\) is non-singular and symmetric. For \(x \in \mathbb{F}\), we let \(\eta(x) = x^{(|\mathbb{F}| - 1)/2}\) denote its Legendre symbol (extended to include 0).

Our starting point is the analysis in [97], which gives a sequential root counting algorithm for general quadratic forms. In fact, [97] also yields a \textit{parallel} algorithm for root counting of \textit{non-singular} quadratic forms, as implied by the following lemma.

**Lemma 4.5.** [97, Theorem 6.27] Let \(Q \in \mathbb{F}^{n \times n}\) be a symmetric quadratic form of full rank over a field \(\mathbb{F}\) of odd characteristic \(p\). Let \(k = \lfloor n/2 \rfloor\); let \(v(b) = |\mathbb{F}| - 1\) for \(b = 0\), and \(v(b) = -1\) for \(b \neq 0\). Then the signature of \(Q\) is as follows:

\[
87
\]
\[ \text{sig}_Q(b) = |\mathbb{F}|^{-1} + v(b) \cdot |\mathbb{F}|^{-(k+1)} \cdot \eta((-1)^k \cdot \det(Q)) \text{ if } n \text{ is even.} \]

\[ \text{sig}_Q(b) = |\mathbb{F}|^{-1} + |\mathbb{F}|^{-(k+1)} \cdot \eta((-1)^k b \cdot \det(Q)) \text{ if } n \text{ is odd.} \]

The above formula is computable in \( \text{NC}^2 \) for fixed characteristic \( p \) (or given an oracle to quadratic residuosity over \( \mathbb{F}_p \) for fields \( \mathbb{F} \) of a non-constant characteristic \( p \)).

We now reduce the case of general quadratic forms to the case of non-singular quadratic forms. A trivial special case is given by the following observation.

**Observation 4.3.** Let \( Q \in \mathbb{F}^{n \times n} \) be a canonical form of rank \( t \), where \( Q = \text{blocks}(Q', 0) \). Then \( \text{sig}_Q = \text{sig}_{Q'} \).

The following key lemma gives a parallel algorithm for transforming an arbitrary symmetric quadratic form \( Q \) into an equivalent canonical form.

**Lemma 4.6.** Let \( Q \in \mathbb{F}^{n \times n} \), where \( Q \) is symmetric of rank \( t \), and let \( I \subseteq [n] \) be a set of \( t \) rows constituting a basis of \( Q \)'s row space. Then \( Q_{I,I} \) is of full rank. Furthermore, there is an equivalence transformation \( C \) such that \( Q' = C^T QC \) is of the canonical form \( Q' = \text{blocks}(Q_{I,I}, 0) \), where \( C \) can be found in \( \text{NC}^2 \) given \( Q \).

**Proof.** Let \( C' \) be such that \( C'^T Q \) leaves the rows \( Q_I \) intact, and makes all other rows 0 by adding to each a proper (unique) combination of the rows in \( Q_I \) (this can be done because \( Q_I \) spans the row space of \( Q \)). Note that \( C'^T \) is invertible. By symmetry (in particular, since the set \( I \) of columns spans \( Q \)'s column space), \( C'^T QC' \) is a matrix which agrees with \( Q \) in its \((I, I)\)-entries and contains 0's in its \((I, [n] \setminus I)\). It is 0 in its \(([n] \setminus I, [n])\) entries since \( C'^T Q \) is, and by choice of \( C' \). Thus, letting \( C = C'P \) for a suitable permutation matrix \( P \), the matrix \( Q' = C^T QC \) satisfies \( Q' = \text{blocks}(Q_{I,I}, 0) \) as required.

The fact that \( \text{rank}(Q') = \text{rank}(Q) \) follows from \( \text{rank}(Q_{I,I}) = \text{rank}(Q') = \text{rank}(Q) \), where the latter equality follows from the fact that \( Q' \) is obtained from \( Q \) by multiplying it with non-singular matrices.

Finally, an \( \text{NC}^2 \) algorithm for finding \( C \) may proceed follows.

1. Find a set \( I \) as above using Section 4.2.2, Item 4.

2. Find \( C' \) as above by solving a system of linear equations (Section 4.2.2, Item 1).

3. Let \( P \) be a permutation matrix mapping \( I \) to \([t]\). Return \( C = C'P \).

The above lemmas directly yield a parallel root counting algorithm for arbitrary quadratic forms over fields of odd characteristic. In the next section we extend this to general degree-2 polynomials.

### 4.4.2 General degree-2 polynomials

The following algorithm reduces root counting for general degree-2 polynomials to the task of finding a canonical \( Q' \) equivalent to \( Q \), along with a suitable equivalence transformation \( C \). We refer to the latter task as simplifying quadratic forms.
Construction 4.1.
Input: \( q(x_1, \ldots, x_n) = x^T B x + h^T x + a \) over a finite field \( \mathbb{F} \) of an odd characteristic, where \( B \) is symmetric.
Output: The number of roots of \( q(x) \).

1. Given \( B \), find \((C, B')\) as in Lemma 4.6, and let \( t = \text{rank}(B) \), \( B'' = B'_{[t], [t]} \).
2. Define \( q'(x) = q(Cx) \). If \( q'(x) \) contains a variable \( x_i \) for \( i > t \), output \( |\mathbb{F}|^{n-1} \).
3. Solve \( d^T B' = -h^T C / 2 \) for \( d \). Let \( u = d^T B'd + h^T C d + a \). Output \( |\mathbb{F}|^n \cdot \text{sig}_{B''}(-u) \).

Claim 4.5. Construction 4.1 is an NC^2 reduction from root counting for general degree-2 polynomials over \( \mathbb{F} \), to quadratic residuosity over \( \mathbb{F} \).

The high-level idea is to find an (injective) affine transformation \( T \) on the input variables, such that \( q(T(x)) \) is a degree-2 polynomial for which root counting easily reduces to root counting for its quadratic part. The transformation \( T \) is found as a composition of (up to) two transformations.

A key observation is the following.

Observation 4.4. The quadratic part of \( q'(x) = q(Cx) \) is equal to \( B' \).

Proof. Substituting \( Cx \) into \( q(x) \), we get

\[
q'(x) = (Cx)^T B(Cx) + h^T Cx + a
= x^T (C^T B C)x + h^T Cx + a
= z^T B'z + h^T Cx + a,
\]

as claimed. \qed

Proof. (of Claim 4.5) The complexity of the reduction follows from the algorithm’s description and the parallel algorithms described in Section 4.2.2. As to correctness, since equivalence transformations preserve signatures, it is sufficient to consider the signature of \( q'(y) \). Now, \( B' \) is the quadratic part of \( q' \) by Observation 4.4. By construction, \( B' \) is canonical. If the condition in 2 holds, then \( q'(x) \) is of the form \( q'(x) = f(x_{[n]\{w\}}) + a y_w \) for a non-zero \( a \), so its output distribution is clearly uniform, and the output is correct. Otherwise, denote \( q''(x) = q'(x + d) \). Indeed, \( d \) exists since \( B'' \) is non-singular. So, \( d_{[t]} \) is uniquely determined, and the other coordinates of \( d \) can be arbitrary values. The latter is true since the right hand side, \( h' = -h^T C / 2 \), satisfies \( h'_{[n]\{t\}} = 0 \) (as well as all coefficients of the \( d_i \)'s in the corresponding equations).

Again, \( \text{sig}_{q''} = \text{sig}_{q'} \) since the transformation \( y = x + d \) is a permutation over \( \mathbb{F}^n \). We have

\[
q''(x) = (x + d)^T B'(x + d) + h^T C(x + d) + a
= x^T B'x + 2d^T B'x + d^T B'd + h^T Cx + h^T Cd + a
= x^T B'x + d^T B'd + h^T Cd + a.
\]

Here, the second equality holds since \( B' \) is symmetric. The last equality is by the choice of \( d \) (the linear part cancels out). Clearly, the number of roots of \( q''(x) \) (and thus of \( q(x) \)) equals the number of solutions to \( b'(x) = -u \). As noted in Observation 4.3, the fraction of the solutions to \( b'(x) = -u \) is the same as that of \( b''(x) = u \), so the number of solutions to \( b'(x) = -u \) \((q''(x) = 0)\) is indeed \( |\mathbb{F}|^n \cdot \text{sig}_{b''}(-u) \). \qed
4.5 Root counting and randomizing polynomials

In this section, we consider efficient randomized encodings of functions by RP (Definition 2.3) with perfect privacy and statistical (not necessarily perfect) correctness. We let $\text{QRP}_F$ denote the class of functions $f : \mathbb{F}^n \rightarrow V$ admitting a polynomial-size representation by such RP, in which the degree in $x$ is constant and the degree in $r$ is $2.\footnote{Unlike in the definition of efficient randomizing encodings in Section 2.2, we do not make a-priori requirements of efficient decoding.}$ We will also refer to a relaxed notion of correctness which only requires that the distributions $p(x, r)$ and $p(x', r')$ be distinct (and perfect privacy). If the relaxed notion holds we say that $f$ is weakly represented by $p$.

We show that any function $f \in \text{QRP}_F$ with polynomial-size range has a non-uniform NC algorithm given oracle access to quadratic residuosity modulo the characteristic of $F$. (No oracle access is necessary when the characteristic is constant, or even polynomial in $n$.) Furthermore, this result applies even to weak representation by RP.

Our starting point is the following lemma, implicit in [73].

**Lemma 4.7.** Let $F = \mathbb{F}_{p^\ell}$, and suppose that $f : \mathbb{F}^n \rightarrow \{0, 1\}$ admits a polynomial-size (weak) randomizing polynomials representation as in Definition 2.3, with degree $d_r$ in $r$ and degree $d_x$ in $x$. Then $f$ can be weakly represented by a single polynomial $p(x, r)$ over $\mathbb{F}_p$ with the same amount of randomness and degree restrictions.

In a nutshell, the proof views the randomizing polynomials vector over $\mathbb{F}_{p^\ell}$ as a vector of polynomials over the base field in the standard way (treating the initial variables as $\ell$-tuples of variables over the base field) to reduce the field of representation. Vazirani’s XOR lemma is then applied to obtain a fixed linear combination of the $s$ outputs on which the two output distributions differ.

Lemma 4.7 allows us to reduce the computation of $f$ to comparing between output distributions (signatures) of degree-2 polynomials. Combining this with a parallel root counting algorithm, we get a parallel algorithm for $f$.

**Theorem 4.5.** Suppose $f : \mathbb{F}^n \rightarrow V$ is in $\text{QRP}_F$, where $F$ is a field of fixed characteristic $p$ and $|V| = n^{O(1)}$. Then $f$ is in non-uniform NC$^2$ (resp., NC$^3$) for odd $p$ (resp., $p = 2$). This holds even when $p$ can grow with $n$ given an oracle to quadratic residuosity modulo $p$.

**Proof.** For any pair of values $v, v' \in V$, $f$ restricted to $X = f^{-1}(v) \cup f^{-1}(v')$ $(f_X)$ is a boolean function, and $p(x, r)$ restricted to domain $X$ is an efficient randomizing polynomials representation for it over $\mathbb{F}_{p^\ell}$.

Applying Lemma 4.7 to $f_X$ implies that it can be efficiently represented by a single weak randomizing polynomial $p_X(x, r)$. Given $x \in X$, we should output $v$ if the signature of $p_X(x, r)$ equals that of $p_X(x, r)$ (for some fixed $f(x_v) = v$), and $v'$ otherwise. This, in turn, can be done via a trivial reduction to root counting of degree-2 polynomials over the same field. As $|\mathbb{F}_p^p| = p = n^{O(1)}$, we can simply reconstruct the signature by computing $\text{sig}_{p(x, r)}(b) = \# p(x, r)(0) / p$ for all $b \in \mathbb{F}_p$, and performing pointwise comparison.\footnote{There exists a more efficient reduction from computing signatures to counting roots (in terms of circuit size), suitable also for large fields. The reduction makes only a constant number of calls to the root-evaluation oracle. In a nutshell, it exploits the simple structure of the set of all possible signatures of degree-2 polynomials.}

Denote the resulting circuit (evaluating $f_X$) by $C_{v,v'}$. Now, since the range of $f$ is small ($|V| = n^{O(1)}$), we
can evaluate \( f \) (almost) without changing the circuit depth, and with a multiplicative overhead of \(|V|^2\) in size. This is done by running \( C_{v,v'}(x) \) for all pairs of output values \( v, v' \) on \( x \), and registering the “winner” for each pair. Pick the value \( v \) that “won” in all \(|V| - 1\) executions it participated in.

4.6 Applications to parallel algorithms

In this section we show how to apply the negative result for quadratic randomizing polynomials as a tool for obtaining parallel algorithms. Our approach reduces the task of algorithm design to finding a suitable randomizing polynomials representation.

Recall that, by Theorem 4.5, if \( f : \mathbb{F}^n \to V \) with a small range admits an efficient randomizing polynomials representation as in Definition 2.3, then there exist non-uniform \( \text{NC}^2 \) (\( \text{NC}^3 \)) circuits for \( f \). (In particular, this includes the case of promise problems where \(|V|\) is large, but \( x \) is such that \( f(x) \) belongs to a small, pre-determined subset of the range.)

In the current context it is useful to rely on the fact that we only need a weak representations, in which different output distributions only need to be distinct (rather than statistically far). Another useful observation is that for the purpose of algorithm design, it is sufficient to devise a representation for a function \( f' \) “refining” \( f \), with range \( \bigcup_{v \in V} R_v \), where \(|R_v| = n^{O(1)}\), \( f'(x) \in R_{f(x)} \), and the \( R_v \)'s are disjoint.

The following examples illustrate several problems that admit natural efficient representations by quadratic randomizing polynomials. Using Theorem 4.5, we can get a unified explanation for the existence of parallel algorithms, which were previously shown in very different and ad-hoc ways.

Matrix rank.

**INPUT:** a matrix \( M \in \mathbb{F}^{n \times n} \).

**OUTPUT:** \( \text{rank}(M) \).

**COMPLEXITY:** \( \text{NC}^2 \) [22, 104].

**RANDOMIZING POLYNOMIALS:** \( p(M, (R, S)) = RMS \), where \( R \) and \( S \) are random and independent matrices in \( \mathbb{F}^{n \times n} \).

This representation was suggested and proved perfectly private and statistically correct in [74]. Setting for weak correctness, one can use the single polynomial \( p(M, (r, s)) = r^T Ms \) where \( r, s \) are random in \( \mathbb{F}^n \).

Matrix similarity (promise version).

**INPUT:** a matrix \( M \in \mathbb{F}^{n \times n} \), which is guaranteed to be similar to (exactly) one matrix out of \( M_1, \ldots, M_s \).

**OUTPUT:** \( i \) such that \( M \) is similar to \( M_i \).

**COMPLEXITY (non-promise problem):** \( \text{NC}^2 \) [22, 104].

**RANDOMIZING POLYNOMIALS:** \( p(M, (R, S)) = (RMS, RS) \), where \( R \) and \( S \) are random in \( \mathbb{F}^{n \times n} \).

Privacy follows since for two similar matrices \( Y = T^{-1} XT \) we have a 1-1 mapping \( T : (R, S) \to (RT, T^{-1} S) \) such that \( p(X, (R, S)) = p(Y, T(R, S)) \). On the other hand, correctness is perfect since \( RMS(RS)^{-1} = RMR^{-1} \), which is a representative of \( M \)’s class.

Quadratic residuosity.

**INPUT:** \( x \in \mathbb{F}' \), where \( \mathbb{F}' \) is a degree-\( n \) extension of a field \( \mathbb{F} \) of a constant odd characteristic (e.g., \( \mathbb{F}' = \mathbb{F}_{3^n} \)).

**OUTPUT:** The Legendre symbol of \( x \) (that is, 1 for quadratic residue, \(-1\) for quadratic non-residue, and 0
Input: a matrix \( M \in \mathbb{F}^{n \times n} \).

OUTPUT: \( \det(M) \).

COMPLEXITY: \( \mathrm{NC}^2 \) [104].

We can obtain a parallel algorithm for the determinant over fields of a small characteristic via a parallel reduction to \( f_i \) as defined above: indeed, for a non-singular \( M \) with a unique LDU decomposition, we have \( \det(M) = \prod_{i=1}^{n} f_i(M) \). To compute the determinant of an arbitrary matrix \( M \), we use Lipton’s worst-case to average-case reduction [102]. For any fixed matrix \( M \) and for a uniform choice of a matrix \( R \), computing \( \det(M) \) reduces to evaluating the degree-\( n \) univariate polynomial \( dt(x) = \det(M + xR) \) at \( n + 1 \) fixed non-zero points \( c_1, \ldots, c_{n+1} \in \mathbb{F} \) and interpolating \( dt(0) = \det(M) \). Using \( \mathbb{F} \) of size \( \text{poly}(n) \), the probability that some matrix \( M + c_i R \) is singular, or does not have a unique LDU decomposition vanishes with \( n \). The requirement on the field size can be enforced by using a sufficiently large extension field.

### 4.7 Root counting for characteristic 2

In this section we prove Theorem 4.4 for the case of characteristic 2. As the proof is quite technically involved, we start with some intuition. Recall that our high-level approach in all cases is to reduce counting roots of a general polynomial \( p(x) \) to counting solutions of equations of the form \( q(x) = b \) for quadratic forms \( q(x) \). In the case of \( \mathbb{F}_2 \), this reduction is particularly easy (by observing that \( x^2 = x \) holds over \( \mathbb{F}_2 \)). In the following, we focus on the latter task for the general case of characteristic-2 fields (this task is not much easier for the case of \( \mathbb{F}_2 \)).

We single out a class of forms \( q \), for which computing \( \#_q(b) \) (equivalently, determining its signature) is easy. These forms are also canonical in the sense that every form is equivalent to a form of this class. More specifically, the following types of forms, which are block-diagonal with block size 2, are canonical:

- **Type 0**: let \( q(x_1, \ldots, x_n) = \sum_{i \leq k} x_{2i-1} x_{2i} + a_i x_{2i-1}^2 + b_i x_{2i}^2 \) for some \( k \) such that \( 2k \leq n \), where \( \text{Tr}_{\mathbb{F}_2}(b_i) = 1 \), and \( a_i \in \{0, 1\} \). We define \( \text{Arf}(q) \triangleq \sum_i a_i \).
  
  - If \( \text{Arf}(q) = 0 \), we say that \( q \) is of type 0.0.
  
  - Otherwise (\( \text{Arf}(q) = 1 \)), we say that it is of type 0.1.

- **Type 1**: \( q(x_1, \ldots, x_n) = \sum_{i \leq k} x_{2i-1} x_{2i} + x_{2k+1}^2 \) for some \( k \) such that \( 2k < n \).

It is known that the pair of parameters \((k, \text{type})\) of a canonical form uniquely determines its signature, and this mapping is 1-1 [5]. Also, \( \#_q(a) \) is easy to compute for canonical forms \( Q \). Thus, we can speak of the \((k, \text{type})\) parameters of any form \( Q \) as the parameters corresponding to (all) canonical forms equivalent to \( Q \). In particular, we extend the definition of \( \text{Arf}(Q) \) to arbitrary quadratic forms. In fact, we devise a slightly enhanced algorithm that, simplifies canonical forms (as in Section 4.4), except that canonical forms are defined differently over characteristic 2. The transformation \( C \) will later help in counting roots of general

\footnote{A subset of canonical forms as below comes up in the sequential algorithm for counting \( \#_q(b) \) of a quadratic form \( q \) implicit in [97], which also proceeds by finding a canonical form equivalent to a given quadratic form \( q \). Our main contribution is in devising a parallel algorithm for finding such a form (and an equivalence transformation leading to it).}

\footnote{This is in fact a special case of the Arf invariant defined in [5]. As was first proved in [5], this quantity is indeed invariant under equivalence transformations (but we will rely on a different proof for this fact).}
degree-2 polynomials. To say some more on finding $C$, we reduce all cases to the case of type-0 regular ($n = 2k$) forms. Then, given a type-0 form, we find $C$ employing a “divide and conquer” approach.\textsuperscript{11} Roughly, given such $Q$ we find $C$ such that $C^TQC$ is block-diagonal, having two blocks of equal size, each of which is regular of type 0. We then proceed recursively to obtain a block-diagonal canonical matrix (requiring $\log n$ iterations). To find such a $C$ as needed in a single iteration in $\text{NC}^2$, we solve a suitable affine system. However, to ensure the system has a solution, we first find an equivalence transformation $C'$ such that $q'(x) = q(C'(x))$ corresponds to $S' = Q' + Q'T$, where $S'_{[n/2],[n/2]}$ is non-singular, and proceed with $Q'$ (which has the same signature as $Q$). It turns out that a random $C'$ satisfies the above requirements with constant probability ($\geq 0.014$).\textsuperscript{12} Thus (hiding many of the details), multiplying the $C$'s obtained in the various iterations, we obtain randomized $\text{NC}^3$ circuits for finding an equivalence transformation $C$ such that $C^TQC = Q'' + A$, where $Q''$ is canonical and $A$ is alternate.

In Section 4.7.1 we present some notation used in this Section. Section 4.7.2 presents a root-counting algorithm for quadratic forms, and Section 4.7.3 presents a root-counting algorithm for general polynomials, building on Section 4.7.2.

### 4.7.1 Notation and background

In this section, when addressing the $Q$ corresponding to a quadratic form $q(x)$, we refer to the (unique) corresponding upper triangular matrix, unless stated otherwise.\textsuperscript{13}

For a quadratic form $Q \in \mathbb{F}^{n\times n}_2$, define $K_Q = \{ x | (Q^T + Q)x = 0 \}$, and $S_Q = K_Q \cap \{ x | x^TQx = 0 \}$. A final related notion we will occasionally need is that of a quadratic space. Given a vector space $V \subseteq \mathbb{F}^n$, and a quadratic form $Q \in \mathbb{F}^{n\times n}$, we say that $(V,q)$ is a quadratic space, where $q(x) = x^TQx$ is viewed as a function from $V$ to $\mathbb{F}$. We say that two quadratic spaces $(V_1,q_1), (V_2,q_2)$ are equivalent if there exists a non-singular and onto linear map $c : V_1 \rightarrow V_2$ such that $q_2(c(x)) = q_1(x)$ for all $x \in V_1$. The notion of a signature naturally extends to quadratic spaces, and is taken over $V$ (rather than the entire $\mathbb{F}^n$). We will rely on some algebra (both known and new) presented in Section 2.5.

### 4.7.2 A root-counting algorithm for quadratic forms

As observed above, over $\mathbb{F}_2$ there exists a simple reduction from root counting of general degree-2 polynomials to counting solutions to equations of the form $q(x) = b$ of quadratic forms. In the following, we provide a parallel algorithm for the latter task over general fields of characteristic 2. We further reduce this task to simplifying quadratic forms, which requires the bulk of technical work, and is presented next. We first show how to handle regular $Q$ of type 0, and then show how to reduce the case of general $Q$ to the former case.

\textsuperscript{11}This is equivalent to computing the Arf invariant for such forms, which may be of independent interest. This is also the computationally hardest part of the algorithm, working in $\text{NC}^2$.

\textsuperscript{12}Proving that such a $C'$ exists, turned out to be the most technically involved part. In particular, it requires analyzing the rank distribution of random alternate matrices over $\mathbb{F}_2$.

\textsuperscript{13}Observe, however, that matrices not of this form can emerge in calculations.
4.7.2.1 Simplification of $Q$ with $n = 2k$

On a high level (hiding some of the details), we employ a “divide and conquer” approach. Starting with a regular $Q \in \mathbb{F}^{n \times n}$ of type 0, we find (in parallel) an equivalent block-diagonal $Q'$, consisting of two blocks $Q_1, Q_2$, where each corresponds to a regular quadratic form of type 0 and size approximately $n/2$. We then proceed in the same manner recursively on $Q_1, Q_2$, until obtaining blocks of constant size, for which a canonical form is found as in [97]. Combining the canonical forms found for $Q_1, Q_2$ results in a canonical $Q'$ as required.

Moving from $Q$ to $Q_1, Q_2$ as above will in fact work for $n$ divisible by 4. To handle this problem, we use the following lemma to reduce the case of general $n = 2k$ to the case of $n$ divisible by 4.

**Lemma 4.8**. Given a quadratic form $Q \in \mathbb{F}_2^{n \times n}$ which is upper triangular, the following task is in NC$^2$. Find an equivalence transformation $C$, such that $Q' + A = C^T QC$, where $A$ is alternate and $Q'$ is block-diagonal with two blocks, with the first being either $Q'_{[2],[2]} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$ or $Q'_{[2],[2]} = \begin{pmatrix} 1 & 1 \\ 0 & b \end{pmatrix}$ with $\text{Tr}_{\mathbb{F}_2^2}(b) = 1$.

**Proof.** In [97], a procedure for finding such a $C$ is given, which is an NC$^1$ reduction (performs $O(1)$ oracle calls on each path) to raising some $x \in \mathbb{F}_2^n$ to the power of some $e = O(2^\ell)$. Every such operation can be done in NC$^2$ using the algorithm referenced in Section 4.2.2, item 2 (the complexity here is in fact in terms of input bit length).

Another procedure we will need for the construction is given by the following Lemma.

**Lemma 4.9**. Let $S \in \mathbb{F}^{n \times n}$ denote a symmetric matrix over a finite field $\mathbb{F}$. Assume $S_{[k],[k]}$ is of full rank. Then there exists a non-singular $C$, such that $C^T SC = \text{blocks}(Q_1, Q_2)$, where $S_1$ is a non-singular in $\mathbb{F}^{k \times k}$ and $S_2$ is of rank $n - k$. Furthermore, such a $C$ can be found in NC$^2$.

**Proof.** Let $C'$ be such that $C'^T S$ leaves the rows $S_{[k]}$ intact, and makes $S_{[n]\setminus[k],[k]}$ 0 by adding to each a proper (unique) combination of the rows of $S_{[k]}$ (this can be done because $S_{[k]}$ is non-singular). Note that $C'^T$ is invertible. The matrix $Q' = C'^T QC'$ agrees with $Q$ in its $([k],[k])$-entries and equals 0 in its $([n]\setminus[k],[k])$ entries, as $Q'_{[n],[k]} = (C'^T C)'$. Also, by symmetry (in particular, since the set [k] of columns spans Q’s column space), $Q'$ contains 0’s in its $([k],[n]\setminus[k])$ entries. Since $C'$ is non-singular, $\text{rank}(Q') = \text{rank}(Q)$. So, since $\text{rank}(Q'_{[k],[k]}) = \text{rank}(Q_{[k],[k]}) = Q_{[k],[k]}$, and since $Q'$ is of the form blocks($Q_{[k],[k]}$, $Q_2$), we must have $\text{rank}(Q_2) = \text{rank}(Q) - k$. Finally, $C'$ can be found in NC$^2$, since we merely solve a system of linear equations (see Section 4.2.2, Item 1).

The complete solution for finding $C$ given a regular, type-0 form $Q$ is by applying the following construction to $Q \in \mathbb{F}^{n \times n}$, with $m = \Theta(n)$.

**Construction 4.2**

- **Input:** a type-0 regular ($n = 2k$) upper-triangular quadratic form $Q \in \mathbb{F}_2^{n \times n}$, $m$ is a parameter determining the failure probability of the algorithm.

- **Output:** $C$, where $C^T QC = A + Q'$, $C$ is an equivalence transformation, $Q'$ is canonical (upper triangular), and $A$ is alternate.

---

As follows from Lemma 4.2, the notions “type-0 $Q$ with $2k = n$” and regular $Q$ of type 0” are indeed equivalent.
1. If \( n < N_\ell \), for \( N_\ell \) as in Theorem 4.3, find \( C' \) transforming \( Q \) into a block-diagonal form \( Q' \) comprised of two blocks \( Q'_{[2],[2]}, Q'_{[3,...,n],[3,...,n]} \) as guaranteed by Lemma 4.8. Proceed recursively on \( Q_{[3,...,n],[3,...,n]} \). Combine \( C' \), with the \( C \) found for \( Q'_{[3,...,n],[3,...,n]} \) to obtain \( C'' \) such that \( C''^T Q C'' = A + Q'' \) for a canonical \( Q'' \) and alternate \( A \) (similarly to 2). Return \( C'' \).

2. If \( n \equiv 2 \pmod{4} \), apply the algorithm from Lemma 4.8 to \( Q \), to obtain \( C' \) transforming \( Q \) into \( Q' + A = C'^T Q C' \). Run recursively on \( Q_{[n]\setminus[2],[n]\setminus[2]} \) to obtain \( C'' \). Let \( C = C' \cdot blocks\left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, C''\right) \), return \( C \).

3. Otherwise \( (n \equiv 0 \pmod{4}) \), let \( S = Q + Q^T \). Run \( m \) executions of the following in parallel. Pick a random matrix \( C \in \mathbb{F}^{n \times n} \), and check whether \( C \), and \( (C^T S C)_{[k],[k]} \) are non-singular.
   
   (a) If the condition is satisfied in none of the executions, output “failure”.

   (b) Otherwise, pick the first execution in which it holds, and let \( Q' + A = C^T Q C \).
      
      i. Let \( S' = Q' + Q'^T \). Apply Lemma 4.9 to \( S' \) to obtain a matrix \( C \) such that \( C^T S'C \) is block-diagonal with two blocks \( S'_1, S'_2 \) of size exactly \( k \) each. Let, \( Q_1, Q_2 \) be (upper-triangular) forms such that \( Q_1 + Q_1^T = S'_1 \) (similarly for \( S'_2 \)).
      
      ii. Make 2 recursive calls (in parallel) with the same \( m \) on \( Q_1, Q_2 \), and obtain their corresponding \( C'_i \). Extend each \( C'_i \) to leave the other block the same, and multiply them to get a \( C' \), transforming \( Q \) into canonical form. Return \( C' \).

**Theorem 4.6.** Construction 4.2 is an RNC\(^3\) algorithm for simplifying regular type-0 quadratic forms.

**Proof.** Choosing \( m = \Theta(n) \), the complexity of the algorithm is as claimed since step 2 (solving a linear equation system) can be done in NC\(^2\) [104]. To prove that the above algorithm is correct it suffices to prove that:

1. The algorithm does not return “failure” in any of the iterations with overwhelming probability in \( n \) if called with \( m = \Theta(n) \).

2. Both \( Q_1, Q_2 \) found in 3(b)i are regular of even size assuming \( Q \) is (so they satisfy the prerequisite of the construction, and the recursive call in 3(b)i can be made).

To see item 1 holds, consider a node in the recursion in stage 3. By Lemma 4.1, \( C_1^T Q C_1 = Q' + A \) for some non-singular \( C_1 \), where \( Q' \) is canonical (and alternating \( A \)). Fix these \( C_1, Q' \), and denote \( S' = Q' + Q'^T \). If \( C = C_1 C' \) such that \( C \) is non-singular and \( (C^T S C)_{[k],[k]} \) is non-singular is picked in some iteration, then “failure” is not returned in (a) for this call. Let us analyze the probability of this event in a single iteration. Let \( C' = \begin{pmatrix} L_1 & L_2 \\ L_3 & L_4 \end{pmatrix} \) be a representation of \( C' \) as a \( 2 \times 2 \) block matrix (all blocks are of size \( k \times k \)). Since \( C_1^T S C_1 = S' \) is block diagonal with two identical blocks \( S'_{[k],[k]} \), we have \( C^T S C_{[k],[k]} = C'^T S'C' = L_1^T S_{[k],[k]} L_1 + L_3^T S_{[k],[k]} L_3 \). Now, observe the latter expression is distributed exactly as in Theorem 4.3, so both \( C^T S C_{[k],[k]} \) and \( L_1 \) are non-singular with probability at least 0.014 (over the choice of \( L_1, L_2 \)).

Now, as \( L_1 \) has rank \( k \) (maximal possible), \( C' \) is non-singular with probability \( \leq 0.288 \) (by standard analysis, this is a bound on the probability to complete \( L_3, L_4 \) so that \( C' \) ends up non-singular). Overall we have success
probability \( \geq 0.014 \cdot 0.288 \geq 0.004 \) for a single iteration. Thus, the probability to output “failure” at that node is \( 0.996^m \). Taking union bound over \( \Theta(n) \) recursive calls, we have failure probability \( \leq ne^{\Omega(-m)} \), which is negligible in \( n \). As to item 2, by the structure of \( Q \) (block diagonal with blocks \( Q_1, Q_2 \)), if \( Q_1 \) or \( Q_2 \) was not regular, then we would have \( \text{rank}(S'_1) + \text{rank}(S'_2) = \text{rank}(S') < n \), contradicting the fact that \( Q \) is regular.

\[ \square \]

### 4.7.2.2 Simplification of general \( Q \)

We proceed by a reduction of a regular type-0 \( Q \), and use Construction 4.2.

**Construction 4.3.** Input: A quadratic form \( Q \in \mathbb{F}_2^{n \times n} \).

Output: \( C \), where \( C^T QC = A + Q' \), where \( C \) is an equivalence transformation, \( Q' \) is canonical (upper triangular), and \( A \) is alternate.

1. Find \( K_Q \), specified as a basis \( U_K = \{ u_1, \ldots, u_{n-2k} \} \).
2. Complement \( K_Q \) into a basis of \( \mathbb{F}^n \), obtaining \( U_D \) (when needed interpret \( U_D \) as the matrix \( (u_1| \ldots | u_{2k}) \)).
3. Let \( C'_D \) denote the output of Construction 4.2 applied to \( H = U_D^T Q U_D \).
4. Check whether the quadratic form

\[
p(y_1, \ldots, y_t) = y^T U_K^T Q U_K y
\]

is identically 0 (iff all coefficient are 0), in particular, this is the case if \( K_Q \) is empty. Let

\[
V_1 = \{ u_i \in U_K | q(u_i) = 0 \}
\]

\[
V_2 = U_K \setminus V_1 = \{ u_j, \ldots, u_{j_w} \}.
\]

Let \( U_S = V_1 \cup \{ q(u_{j_a-1})^{1/2} u_{j_a} - q(u_{j_a})^{1/2} u_{j_a-1} \}_{a=2}^w \) be a basis for \( S_Q \).

(a) If it is not, let \( r \) be a vector complementing \( U_S \) into a basis of \( K_Q \). Output \( C = (U_D C'_D | r | U_S) \).
(b) If it is, output \( C = (U_D C'_D | U_S) \).

**Lemma 4.10.** Construction 4.3 is an \( \text{RNC}^3 \) algorithm for computing the canonical form of a quadratic form \( Q \in \mathbb{F}_2^{n \times n} \).

**Proof.** We first prove that \( H \) is regular and of type 0 (so Construction 4.2 can be applied to it). The quadratic space \( (\text{span}(U_D), q) \) is equivalent to \( (\mathbb{R}^{2k'}, h) \) by Observation 4.2. 1. If \( H \) was non-regular with \( 2k' < 2k \), we would have \( \dim(K_H) = 2k - 2k' > 0 \). By the equivalence, it follows that there is a subspace \( \tilde{D} \) of \( \text{span}(U_D) \) of dimension \( 2k - 2k' > 0 \) such that \( \tilde{D} \subseteq K_Q \). However, \( U_D \) was selected in a way that guarantees \( \text{span}(U_D) \cap K_Q = \{ 0 \} \), leading to a contradiction, so \( H \) must be regular. Since \( 2k \) is even, \( H \) is indeed of type 0. If \( Q \) is of type 0, by Corollary 4.1 we have \( K_Q = S_Q \), and \( C \) is as claimed (in particular, it has full rank). Otherwise (\( Q \) of type 1), it remains only to prove that \( U_S \) indeed spans \( S_Q \). Clearly, \( V_1 \in S_Q \), and contains independent vectors. The other vectors clearly complement it into an independent set. To
Here, the last equality follows since $u_{j_{a - 1}}$ is non-zero, by definition of $V_2$. We have

$$q(v) = q(u_{j_{a - 1}})q(u_{j_a}) + (q(u_{j_{a - 1}})^{1/2}u_{j_a})^TQq(u_{j_a})^{1/2}u_{j_{a - 1}} + (q(u_{j_a})^{1/2}u_{j_{a - 1}})^TQq(u_{j_{a - 1}})^{1/2}u_{j_a}$$

$$= (q(u_{j_{a - 1}})^{1/2}u_{j_a})^T(Q + Q^T)q(u_{j_a})^{1/2}u_{j_{a - 1}}$$

$$= 0$$

Here, the last equality follows since $u_{j_{a - 1}} \notin K_Q$. The only non-obvious complexity issue is that $x^{1/2}$ (over $\mathbb{F}$) is computable in NC$^2$. However, since $y^2$ is a linearized polynomial, we can solve $y^2 = x$ in NC$^1$, as explained in Section 4.3.1.

Given a quadratic form $Q \in \mathbb{F}_2^{n \times n}$ we can find an equivalent canonical $Q'$ using Construction 4.3, and count its roots as in Lemma 4.5. For general degree-2 polynomials $p$ over $\mathbb{F}_2$, by eliminating the linear part ($x^2 = x$ over $\mathbb{F}_2$), we obtain some $q(x) + a$, where $q$ is a quadratic form. Proceeding similarly to obtain a canonical $q'$ equivalent to $q$, the number of $p$'s roots equals $2^n \text{sig}_{q'}(a)$.

### 4.7.3 A root counting algorithm for general polynomials

Our algorithm for finding the signature of degree-2 polynomials over $\mathbb{F}_2^k$ proceeds in 2 steps. As before, given a polynomial $p$, we first apply Construction 4.3 to simplify its quadratic part $Q$, obtaining $(C, Q')$. Then, we show how to count the roots of $p$ given $(C, Q')$.

We will need a lemma on signatures of degree-2 polynomials of an additional kind, complementing Lemma 4.1 on the signatures of quadratic forms.

**Lemma 4.11.** Let $q(x_1, \ldots, x_n) = b(x) + x_{2k+1}^2 + ax_{2k+1}$, where $b(x_1, \ldots, x_n)$ is canonical for type 0, and $2k \leq n$, and $a \neq 0$. Let $I$ denote the image of the linear (over $\mathbb{F}_2$) mapping $r(x) = x^2 + ax$ (see Paragraph 4.3.1). Then,

$$\text{sig}_{B}(r) = \begin{cases} 
2^{-\ell} + (1 - 2\text{Arf}(B))2^{-\ell(k+1)} & \text{for } r \in I, \text{ and} \\
2^{-\ell} - (1 - 2\text{Arf}(B))2^{-\ell(k+1)} & \text{otherwise}.
\end{cases}$$

**Proof.** The mapping $r(x)$ has 2 distinct roots 0, $a$. Since it is linear over $\mathbb{F}_2$, its kernel is precisely $\text{span}(\{a\})$ (over $\mathbb{F}_2$), so it has dimension 1. Thus, its image $I$ has dimension $\ell - 1$. Thus $q(x) \in I$ iff $b(x_1, \ldots, x_{2k}) \in I$, furthermore, the conditional output distributions of $q(x)$ over $I$ and over $I + a$ are uniform, since each $b(x_1, \ldots, x_{2k})$ contributes a uniform distribution over the coset $b(x) + r(x_{2k+1})$, since $r(x_{2k+1})$ is uniform over $I$. Based on Lemma 4.1 we get $\Pr[q(x) \in I] = \Pr[q(x) = 0] + (2^{\ell - 1} - 1)\Pr[q(x) = 1] = 2^{-1} + (1 - 2\text{Arf}(B))2^{-\ell(k+1)}$. The result then follows based on the above observation on uniform distributions over cosets.

**Remark 4.3.** Compared with the odd characteristic case, we have an additional “base case” described in Lemma 4.11 in the set of polynomials to which general polynomials are reduced. More specifically, for odd characteristic we are able to either fully eliminate the linear part which shares variables with the bilinear part, so that the signature is preserved. Here a 1-variable intersection is generally inevitable. As a sanity
check, the elimination process as for the odd characteristic does not work for characteristic 2 would not work, since we cannot divide by 2.

**Construction 4.4** Input: a degree-2 polynomial $q(x) = \sum_{i\leq j} a_{i,j}x_i x_j + \sum_{i\in[n]} a_i x_i + a \in \mathbb{F}_{2^\ell}[x_1, \ldots, x_n]$. Output: The number of $q$’s roots.

1. Let $b(x) = \sum_{1 \leq i, j \leq n} a_{i,j} x_i x_j$ denote the quadratic part of $q$. Run Construction 4.3 on $b$ to obtain an equivalence transformation $C$ to a canonical form $B$. Let $q_1(x) = q(Cx) = \sum_{i \leq k} a_{i,i}^\prime x_{2i-1} x_{2i} + \sum_{i \leq k} a_i^\prime x_i + a^\prime$, and let $b_1(x)$ be the quadratic part of $q_1(x)$ ($B_1$ is canonical by correctness of Algorithm 4.3).

2. If $q_1(x)$ contains variables not appearing in $b_1(x)$, return $2^{\ell(n-1)}$.

3. Otherwise, $d = \sum_{j=1}^k (a_{i,j}^\prime a_{j-1}^\prime + a_{2j-1}^\prime + a_{2j})$ (for the $k$ corresponding to $B_1$).
   - (a) If $B_1$ is of type 0, or if $a_{2k+1}^\prime = 0$, output $2^{\ell n} \text{sig}_{b_1}(a^\prime + d)$.
   - (b) Otherwise, return $\text{sig}_{q_2}(a^\prime + d) 2^{\ell n}$, where $q_2(x) = b_1(x) + a_{2k+1}^\prime x_{2k+1}$. To test membership in $I$, solve the equation $x_{2k+1}^2 + a_{2k+1}^\prime x_{2k+1} = a^\prime + d$ as explained in Section 4.3.1.

**Theorem 4.7.** Construction 4.2 is an RNC$^3$ algorithm for counting roots of degree-2 polynomials over $\mathbb{F}_{2^\ell}$.

**Proof.** As usual, the non-singular transformation of the input variables $x \to Cx$ in 1, does not change the signature of $q$ (since it is a permutation of $\mathbb{F}^n$).

- In 2 we cover the case when $q_1$ has linear terms not appearing in the quadratic part, thus its signature is clearly uniform over $\mathbb{F}_{2^\ell}$.
- In 3.a we cover the case when $q_1(x) = \sum_{i \leq k} a_{i,i}^\prime x_{2i-1} x_{2i} + c x_{2k+1}^2 + \sum_{i \leq k} a_i^\prime x_i + a^\prime$, for $c \in \{0, 1\}$. Substituting $x_{2i-1} \to x_{2i-1} + a_{2i}$, and $x_{2i} \to x_{2i} + a_{2i-1}^\prime$ for $i \leq k$, we obtain a polynomial $q_2'(x) = b_1(x) + a^\prime + d$, having the same signature as $q_1$, (as the substitution function is a permutation of $\mathbb{F}^n$). We know how to address $q_2'$’s root counting by Lemma 4.1.
- In 3.2 we cover the case of $q_1(x) = \sum_{i \leq k} a_{i,i}^\prime x_{2i-1} x_{2i} + x_{2k+1}^2 + \sum_{i \leq k} a_i^\prime x_i + c x_{2k+1} + a^\prime$ for $c \neq 0$. Applying the same transformation as in the previous item, we obtain a polynomial $q_2'(x) = b_1(x) + a_{2k+1}^\prime x_{2k+1} + a^\prime + d$. $q_2'$ matches the conditions of Lemma 4.11 (up to an additive constant), so we know how to address its root counting.

As the above cases cover all possibilities, we are done. 

\[\square\]
Chapter 5

Evaluating branching programs on encrypted data

In this chapter, we present our results on computing on encrypted data.

5.1 Introduction

Computing on encrypted data is arguably one of the most intriguing open problems in cryptography. The variant of this problem we are interested in may be illustrated by the following motivating scenario. Suppose that a client, holding a sensitive local input $x$, wishes to run a remote program $P$ on this input. For instance, $x$ can be the medical history of an individual and $P$ a complex propriety algorithm determining whether to offer insurance coverage to this individual. To the end of evaluating $P(x)$, the client wishes to publish an encrypted version of $x$, denoted by $c$, while still allowing a server owning $P$ to effectively run its program on the ciphertext $c$. That is, based on $P$ and $c$ the server should compute in polynomial time a message $c'$, from which the client can recover $P(x)$ using its secret key.

As described so far, the problem can be solved by simply letting $c'$ include a complete description of $P$. However, this trivial solution has two significant weaknesses. First, it completely reveals $P$ to the client, whereas ideally the client should only be able to learn $P(x)$. Second, when the description size of $P$ is bigger than its input and output, this solution is wasteful in terms of communication. Ideally, the communication should be a-priori bounded by some polynomial in the size of the input $x$, the output $P(x)$ and the security parameter, independently of the description size of $P$. The same holds for the amount of local computation and storage used by the client. To summarize, it is desirable to obtain solutions which satisfy the following two goals:

1. Hide $P$ from the client (to the extent possible).

2. Make the client’s work independent of the size of $P$. In particular, $c'$ should be succinct in the sense that its size depends only on the size of the input and output and not on that of $P$.

Jumping ahead, the main open problem in the area is that of realizing the second goal. This problem is the focus of our work.
Before addressing known methods for realizing the above two goals, it is instructive to further clarify what we mean when referring to a “program” $P$. A program is a string that represents a function, mapping an input $x$ to an output $y$. To simplify the exposition, we restrict the attention to finite boolean functions $f : \{0,1\}^n \rightarrow \{0,1\}$. The correspondence between a program $P$ and the function it represents is determined by an underlying representation model. Common representation models for finite functions include circuits, formulas, branching programs, OBDDs, finite automata, decision trees, and truth tables. Once the representation model is fixed, every string $P$ has a unique interpretation as a program computing some specific function $f$. In this work we will be interested in universal representation models, in which every function $f$ can be computed by some program $P$ in the model. Note that all of the models in the above list are universal. However, the complexity of representing a function can greatly vary between the models. Circuits are the most powerful model in the list, in the sense that a program in any of the other models has an equivalent circuit of essentially the same size. On the other extreme, truth tables are the least powerful of these models, requiring a program of size $2^n$ for any function $f$. This makes truth tables useless for all but very small input lengths $n$.

We return to the question of realizing the above two goals. Goal 1 can be addressed by using techniques from the area of secure computation. Most notably, Yao’s garbled circuit technique [124, 24, 99] can handle any circuit $P$, allowing to computationally hide all information about $P$ other than $P(x)$ and the size of $P$. A similar result can be obtained for less powerful representation models, such as formulas [118, 91, 92] or various kinds of branching programs [9, 49, 73, 116] with the additional feature of keeping $P$ information-theoretically private. (The latter variant of computing on encrypted data was referred to as “cryptocomputing” by Sander et al. [118].) However, all these techniques inherently fail with respect to Goal 2, as they require the size of $c'$ to be comparable to the size of $P$. This gives rise to the following question:

For which natural representation models can we realize Goal 2, namely evaluate an arbitrary program $P$ on an encrypted input so that the client’s work does not depend on the size of $P$?

A positive answer for the case of circuits (hence also for all other models) would easily follow from the existence of a homomorphic encryption scheme — one that allows to freely perform both additions and multiplications on ciphertexts. However, there is yet no candidate for an encryption scheme with this strong property.

The first protocols in which the client’s work can go below the size of $P$ were given in the context of Private Information Retrieval (PIR) [32, 94]. A single-server PIR protocol can be viewed as a protocol for evaluating a truth table $P$ of size $N = 2^n$ on an encrypted input $x$ of size $n$. There are such protocols in which the client’s work is polynomial in $n$ [25, 101], thus affirmatively answering the above question for the case of a truth table representation. Extensions to a set representation (where $P$ lists the set of inputs on which $f$ evaluates to 1) were given in the context of private keyword search [94, 31, 54, 110]. Recently, a cryptosystem which allows to evaluate 2-DNF formulas and degree-2 polynomials on encrypted data was given by Boneh et al. [21].\footnote{In fact, the cryptosystem of [21] realizes a stronger form of computing on encrypted data in which the length of the ciphertext $c'$ depends only on the security parameter and not on the length of the input.} The question of realizing Goal 2 for more powerful and useful representation models remained open.
5.1.1 Our contribution

We obtain an affirmative answer to our main question for the case of length-bounded branching programs. To explain the meaning of this result, we give some background on branching programs and their complexity. A (deterministic) branching program \( P \) is defined by a directed acyclic graph in which the nodes are labeled by input variables and every nonterminal node has two outgoing edges, labeled by 0 and 1. An input \( x \in \{0, 1\}^n \) naturally induces a computation path from a distinguished initial node to a terminal node, whose label determines the output \( P(x) \). The size of \( P \) is defined as the number of nodes in the graph and its length is the length of the longest path from the initial node to a terminal node. Branching programs are a relatively powerful representation model. In particular, any logarithmic space or \( \text{NC}^1 \) computation can be carried out by a family of polynomial-size branching programs.

We consider classes of branching programs whose length is bounded by some public parameter \( \ell \), where \( \ell = \ell(n) \) is polynomial in \( n \). Representation by \( \ell(n) \)-bounded branching programs is universal whenever \( \ell(n) \geq n \). Indeed, any function \( f \) can be computed by a complete decision tree of length \( n \) and size \( O(2^n) \). Branching programs of length \( \ell(n) = n \) are of special interest, as they can simulate several representation models that are often used in practice. For instance, if \( f \) can be computed by a deterministic finite automaton with \( s \) states, then it can be computed by a branching program of length \( n \) and size \( sn + 1 \). Other useful models such as decision trees and OBDDs are also special cases of length-\( n \) branching programs. Finally, efficient data structures for various natural search problems (such as tries and balanced search trees) can be implemented by branching programs with a small length.

Our main result is a public-key encryption scheme with the following properties. Given a branching program \( P \) and an encryption \( c \) of an input \( x \), it is possible to efficiently compute a succinct randomized ciphertext \( c' \) from which \( P(x) \) can be efficiently decoded using the secret key. The size of \( c' \) and the work required for decrypting it depend polynomially on the size of \( x \) and the length of \( P \), but do not further depend on the size of \( P \). Thus, whenever the length \( \ell(n) \) is some fixed (polynomial) function of \( n \), we realize Goal 2 above. As interesting special cases, one can evaluate finite automata, decision trees, and OBDDs on encrypted data, where the size of the resulting ciphertext \( c' \) does not depend on the size of the object being evaluated. These are the first general representation models for which such a feasibility result is shown. We also strengthen the above protocol to realize Goal 1 in a very strong sense, guaranteeing that \( c' \) does not contain additional information about \( P \) (other than \( P(x) \) for some \( x \)) even if the public key and the ciphertext \( c \) are maliciously formed.

**Size hiding.** Our protocols have the following size hiding feature: the ciphertext \( c' \) does not reveal any information whatsoever about the size of \( P \), no matter how large \( P \) is. This should be contrasted with previous methods of computing on encrypted data, in which the communication complexity and the client’s work directly reflect (an upper bound on) the size of \( P \). Thus, we achieve a stronger version of Goal 1 than in all previous solutions. A similar notion of size hiding was previously considered by Micali et al. in the context of zero-knowledge sets [90].

**Applications to secure two-party computation.** Our technique for computing on encrypted data immediately gives rise to a one-round (two-message) secure protocol for evaluating a length-bounded branching

---

\( ^2 \)We note that perfect size hiding cannot be achieved in the physical reality, as the time it takes the server to respond reveals an upper bound on the size of \( P \). However, increasing this upper bound on the size of \( P \) does not involve additional work. This should be contrasted with the partial size hiding that can be achieved using previous protocols by simply padding the inputs.
program $P$ held by a server on an input $x$ held by a client. (In the semi-honest model, this also implies a protocol for the setting in which $P$ is public but its inputs are partitioned between the two parties.) In the case of malicious parties, the protocol satisfies the same relaxed security definition used in previous works on one-round secure computation in the plain model [106, 1, 54, 82, 96]. A distinctive feature of our protocol is that the client’s work is independent of the size of $P$ and moreover the protocol hides the size of $P$ from the client.\footnote{A secure two-party protocol in which the communication complexity is almost independent of the size of $P$ can be obtained using the technique of Naor and Nissim [105]. However, this protocol requires multiple rounds of interaction and does not achieve size hiding.} The latter size hiding feature demonstrates that while hiding the sizes of both inputs is generally impossible, there are useful cases where one can hide the size of one of the inputs while maintaining the secrecy of the other input (but not its size).

As a concrete application, one can obtain a secure one-round protocol for keyword search which totally hides from the client the size of the data set held by the server. That is, a client holding a secret keyword $x$ can query a database $D$ held by a server without revealing $x$ and while assuring the server that it cannot learn anything about $D$ (including its size) other than whether $x \in D$. Previous solutions to the secure keyword search problem [31, 54, 110] fall short of achieving the size hiding goal. A size hiding protocol as above is obtained by representing $D$ as a trie data structure, which can be viewed as an instance of a length-$n$ branching program.

We finally note that the one-round protocol obtained using our technique yields a simpler alternative to similar protocols from the literature that provide unconditional security to the server [118, 9, 73, 116, 92]. Its complexity improves over previous protocols even in the case of branching programs of unbounded length. For evaluating a branching program of size $s$ over $n$ binary inputs, the communication complexity of our protocol is $O(ksn)$ (where $k$ is a security parameter), improving over the $O(ks^2)$ complexity of the best previous solutions in this setting [73]. It should be noted, however, that the latter solutions apply also to non-deterministic branching programs, whereas our technique applies only to the deterministic model.

**Techniques.** The basic version of our protocol uses a simple generalization of the technique of Kushilevitz and Ostrovsky [94] for constructing single-server PIR protocols. In fact, the protocol of [94] (as well as its variants from [122, 101]) can be viewed as an instance of our protocol in which the branching program is a complete (but possibly non-binary) decision tree whose $i$-th level depends only on the $i$-th input variable.

Our protocol proceeds roughly as follows. The ciphertext $c$ is obtained by separately encrypting each bit of $x$ using an additively homomorphic public-key encryption scheme. (For efficiency reasons we rely on the Damgård-Jurik scheme [43]; this scheme was previously used in the context of PIR by Lipmaa [101].) To evaluate $P$ on $x$ we proceed in a bottom up manner. Starting from the terminal nodes, in the $i$-th iteration we handle all nodes whose distance from the terminal nodes is $i$. For each such node, we compute a ciphertext containing an (iterated) encryption of its value. Using the homomorphic property, the encryption assigned to every node can be computed from the encryptions assigned to its children (which were computed in previous iterations) and the encryption of the input bit labeling this node. The ciphertext $c'$ is the (iterated) encryption assigned to the initial node. The client can recover $P(x)$ by applying iterated decryptions to $c'$.

The stronger variant of our protocol which remains secure in the case of malicious clients is more involved, and relies on variants of previous techniques of Aiello et al. [1], Naor and Pinkas [106], Laur and Lipmaa [96], and (especially) Kalai [82].
In Section 2.4 we define our general notion of representation models as well as the specific branching program model for which our results apply. Section 5.2 contains some algebraic claims used in this chapter. In Section 5.3 we define the problem of computing on encrypted data as well as a variant of Oblivious Transfer on which our solution relies. Our main protocol is presented in Section 5.4. This protocol guarantees the privacy of the client as well as the privacy of the server against a semi-honest client. The case of malicious clients is discussed in Section 5.5.

5.2 Preliminaries

In this section we present some simple technical claims used throughout this chapter.

Fact 5.1. [Chinese Remainder Theorem] Consider $N = p_1 p_2$, where $p_1, p_2 > 1$ are coprime. Then:

1. $\mathbb{Z}_N$ is isomorphic to $\mathbb{Z}_{p_1} \times \mathbb{Z}_{p_2}$, via the isomorphism $\eta(x) = (x \mod p_1, x \mod p_2)$.

2. $\mathbb{Z}_N^*$ is isomorphic to $\mathbb{Z}_{p_1}^* \times \mathbb{Z}_{p_2}^*$ via the same isomorphism.

Fact 5.2. [Structure of $\mathbb{Z}_p^*$] For every odd prime $p$ and positive integer $h$, the group $\mathbb{Z}_p^*$ is isomorphic to $\mathbb{Z}_{p^h - 1} \times \mathbb{Z}_{p - 1}$. It follows that $\mathbb{Z}_p^*$ must be cyclic.

Claim 5.1. Let $(T, +)$ be a cyclic group, and let $x, y, \mu \in T$, such that $p = \gcd(\text{ord}_T(x), \text{ord}_T(y))$. Then, if $S = \{ -1 < a < \text{ord}_T(x) \mid \exists -1 < b < \text{ord}_T(y), ax + by = \mu \} \neq \emptyset$, then $|S| = p$. Furthermore, for all $a' \in S$, $|\{ (a, b) | ax + by = \mu, a = a' \}| = 1$.

Proof. The set $S_x = \{ x^0 = 1, x^{\text{ord}(x)/p}, \ldots, x^{(p-1)-\text{ord}(x)/p} \}$ is a subgroup of $T$ of order $p$ (all elements are distinct, or else, $\text{ord}(x)$ would be smaller than it is). Similarly,

$$S_y = \left\{ y^0 = 1, y^{\text{ord}(y)/p}, \ldots, y^{(p-1)-\text{ord}(y)/p} \right\}$$

is a subgroup of $T$ of order $p$. Since a subgroup of a given order of a cyclic group is unique, we must have $S_y = S_x$. Thus, for every $a$ which is a multiple of $\text{ord}(x)/p$, there exists a $b$ which is a multiple of $\text{ord}(y)/p$, such that $xa = -by$, and thus, we have obtained a set of $p$ solutions to $ax + by = 0$, in which all the $a$’s are distinct. To see that these are the only possible solutions to $ax + by = 0$, assume the contrary, and let $(a, b)$ be such that $ax + by = 0$, and, wlog. $ax \notin S_x$. $\text{ord}(ax)$ divides $\text{ord}(x)$, as $\langle ax \rangle$ is a subgroup of $\langle x \rangle$. On the other hand, $\text{ord}(ax)$ does not divide $p$, or else $ax$ would be in $S_x$. Thus, by choice of $p$, $\text{ord}(ax)$ does not divide $\text{ord}(y)$ (or else, $p$ would be different). Thus, there is no solution to $ax + by = 0$, since a solution $(a, b)$ satisfies $\langle ax \rangle$ is a subgroup of $\langle by \rangle$ ($ax = by$), which is in turn a subgroup of $\langle y \rangle$, so we would have $\text{ord}(ax)|\text{ord}(y)$ - a contradiction. Since given a solution $(a', b')$ to $ax + by = \mu$, $\{ (a, b) | ax + by = \mu \} = \{ (a + a', b + b') | ax + by = 0 \}$, the result follows.

Claim 5.2. For $x \in \mathbb{Z}_n^{\ast}$, let $x' = (x \mod N^e)$ be an element of $\mathbb{Z}_n^{\ast}$. Then $\text{ord}(x')$ divides $\text{ord}(x)$.

Proof. Note that $x'^t = 1 \mod N^e$ if and only if $x^t = z N^e + 1 \mod N^{e+1}$ for some $z \in \mathbb{Z}$. This also holds for the minimal integer $d'$, for which $x'^{d'} = 1 \mod N^e$. $\text{ord}(x)$ is then $d = d' \cdot \text{ord}_{\mathbb{Z}_n^{\ast}}(z N^e + 1)$, which is divisible by $d'$. \qed
5.3 Cryptographic primitives

In this section we define both our goal of computing on encrypted data and the main cryptographic building block on which we rely.

5.3.1 Computing on encrypted data

We consider a scenario where a client, holding an input $x$, publishes a public key $pk$ and an encryption $c$ of $x$ under $pk$. This encryption is used by a server to efficiently evaluate a program $P$ (in some given representation model) on $c$, obtaining a ciphertext $c'$. The client then uses its secret key to recover $P(x)$ from $c'$. This is formalized as follows.

Definition 5.1. [Computing on encrypted data] Let $U : \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}^*$ be a representation model. A protocol for evaluating programs from $U$ on encrypted data is defined by a tuple of algorithms $(\text{Gen}, \text{Enc}, \text{Eval}, \text{Dec})$ and proceeds as follows.

- **Setup:** Given a security parameter $k$, the client computes $(pk, sk) \leftarrow \text{Gen}(1^k)$ and saves $sk$ for a later use.
- **Encryption:** The client computes $c \leftarrow \text{Enc}(pk, x)$, where $x$ is the input on which a program $P$ should be evaluated.
- **Evaluation:** Given the public key $pk$, the ciphertext $c$, and a program $P$, the server computes an encrypted output $c' \leftarrow \text{Eval}(1^k, pk, c, P)$.
- **Decryption:** Given the encrypted output $c'$, the client outputs $y \leftarrow \text{Dec}(sk, c')$.

We require that if both parties act according to the above protocol, then for every input $x$, program $P$, and security parameter $k \in \mathbb{N}$, the output $y$ of the final decryption phase is equal to $U(P, x)$ except, perhaps, with negligible probability in $k$.

An essential security requirement for computing on encrypted data is client privacy, requiring that the pair $(pk, c)$ produced in the above process keep the client’s input $x$ semantically secure [65, 60].

Definition 5.2. [Client privacy] Let $\Pi = (\text{Gen}, \text{Enc}, \text{Eval}, \text{Dec})$ be a protocol for computing on encrypted data. We say that $\Pi$ satisfies the client privacy requirement if the advantage of any PPT adversary $\text{Adv}$ in the following game is negligible in the security parameter $k$:

- $\text{Adv}$ is given $1^k$ and generates a pair $x_0, x_1 \in \{0, 1\}^*$ such that $|x_0| = |x_1|$.  
- Let $b \leftarrow \{0, 1\}$, $(pk, sk) \leftarrow \text{Gen}(1^k)$, and $c \leftarrow \text{Enc}(pk, x_b)$.
- $\text{Adv}$ is given the challenge $(pk, c)$ and outputs a guess $b'$.

The advantage of $\text{Adv}$ is defined as $\Pr[b = b'] - 1/2$. 

105
Client privacy alone can be realized by simply letting Eval output $P$. However, it becomes nontrivial to satisfy when $|P| \gg |x|$ and the communication complexity is required to be sublinear in $|P|$. The latter requirement is in the center of this work.

While client privacy suffices for some applications, we will also be interested in protecting the privacy of the server by hiding the program $P$ to the extent possible. For simplicity we consider here the case of a semi-honest client, who generates a valid public key $pk$ and ciphertext $c$. The case of malicious clients will be addressed in Section 5.5.

**Definition 5.3.** [Server privacy: semi-honest model] Let $\Pi = (\text{Gen, Enc, Eval, Dec})$ be a protocol for evaluating programs from a representation model $U$ on encrypted data. We say that $\Pi$ has statistical server privacy in the semi-honest model if there exists a PPT algorithm $\text{Sim}$ and a negligible function $\epsilon(\cdot)$ such that the following holds. For every security parameter $k$, input $x \in \{0, 1\}^*$, pair $(pk, c)$ that can be generated by $\text{Gen, Eval}$ on inputs $k, x$, and program $P \in \{0, 1\}^*$, we have

$$\text{SD}(\text{Eval}(1^k, pk, c, P), \text{Sim}(1^k, 1|x|, pk, U(P, x), 1|P|)) \leq \epsilon(k).$$

The case of perfect server privacy is defined similarly, except that $\epsilon(k) = 0$ and $\text{Sim}$ is allowed to run in expected polynomial time.

In the case of computational server privacy, $\text{Sim}$ should satisfy the following requirement. For every polynomial-size circuit family $D$ there is a negligible function $\epsilon(\cdot)$ such that for any $k, x, pk, c, P$ as above we have

$$\Pr[D(\text{Eval}(1^k, pk, c, P)) = 1] - \Pr[D(\text{Sim}(1^k, 1|x|, pk, U(P, x), 1|P|)) = 1] \leq \epsilon(k).$$

Our main protocol will have perfect server privacy. In fact, it will additionally hide the size of the server’s input $P$ from the client. We refer to this property as size hiding. This implies, in particular, that the length of $c'$ must be independent of the length of $P$.

**Definition 5.4.** [Size hiding server privacy: semi-honest model] We say that $\Pi$ has (perfect, statistical, or computational) size hiding server privacy in the semi-honest model if it satisfies the requirements of Definition 5.3 with the difference that $\text{Sim}$ does not get the length of $P$ as an input.

**Remark 5.1.** (On flexibility of input lengths) The above definitions relax standard definitions of secure computation from the literature (e.g., [27, 60]) in that they do not impose an a-priori polynomial relation between the lengths of the two inputs and the security parameter. This relaxation is motivated by the size hiding variant of the definition, where we allow an arbitrary gap between the length of the two inputs while still requiring the client to run efficiently in the length of its own input. Similar definitions have been used in other contexts of “sublinear-communication cryptography” such as private information retrieval (cf. [101]), communication-efficient arguments (cf. [7]) and zero-knowledge sets [90]. We note that our length-flexible definitions of client privacy (Definition 5.2) and server privacy (Definition 5.3) can be realized by one-round secure computation protocols from the literature. In particular, Yao’s protocol [124] (see also [49, 101])

---

4In the definitions from [60] it is assumed that both inputs have the same length, and the security parameter is taken to be this input length. In the definitions of [27] the running time of the honest parties is bounded by a fixed polynomial in the security parameter, effectively imposing a similar bound on the input length.
yields a client-server protocol for any polynomial-time computable function \( f \) with computational server privacy, and protocols from [118, 91, 49, 73, 9, 92] yield client-server protocols for any function \( f \) in \( \text{NC}^1 \) or even \( \text{NL} \) with perfect or statistical server privacy. None of these protocols satisfies the additional size hiding property.

**Remark 5.2.** Protocols \( \Pi \) which satisfy our definitions of client privacy (Definition 5.2) and standard server privacy (Definition 5.3) can be easily derived from previous protocols for one-round secure computation. In particular, Yao’s protocol [124] yields a protocol for evaluating circuits on encrypted data with computational server privacy, and protocols from [118, 91, 49, 73, 9, 116, 92] yield protocols for evaluating formulas, branching programs, and even non-deterministic branching programs on encrypted data with perfect or statistical server privacy. However, in all these protocols the length of \( c' \) is generally bigger than the length of \( P \). In particular, none of these protocols satisfies the additional size hiding property of Definition 5.4.

**Remark 5.3.** A protocol \( \Pi \) as above can be trivially turned into a one-round secure two-party protocol for the function \( U(\cdot, \cdot) \). In this protocol the client (on input \( x \)) sends \( pk, c \) to the server, the server (on input \( P \)) responds with \( c' \), and the client recovers the output \( U(P, x) \). The protocol satisfies a strong simulation-based security definition which strengthens standard definitions of secure computation from the literature (e.g., [27, 60]) in that it does not impose an a-priori polynomial relation between the lengths of the two inputs and the security parameter.

### 5.3.2 Oblivious transfer

It will be convenient to present our main protocol in a modular way, using a variant of one-round Oblivious Transfer (OT) [114, 48] as a subprotocol. To this end it will be necessary to rely on a stronger server privacy property than the one implied by standard definitions of OT. As before, we focus here on the case of a semi-honest client and postpone the treatment of malicious clients to Section 5.5.

A standard one-round OT protocol involves a server, holding a list of \( t \) secrets \((s_1, s_2, \ldots, s_t)\), and a client, holding a selection index \( i \). The client sends a query \( q \) to the server, who responds with an answer \( a \). Using \( a \) and its random input, the client should be able to recover \( s_i \). The standard security requirements include **client privacy**, requiring that \( q \) keep \( i \) hidden from the server, and **server privacy**, requiring that \( a \) keep all secrets other than \( s_i \) hidden from the client. Note that the latter server privacy requirement does not rule out the possibility that \( a \) reveals information about the query \( q \) which is not implied by the output \( s_i \) alone. (In fact, \( a \) can include the entire query \( q \) without violating server privacy.) This might compromise the security of our main protocol, in which the client issues multiple OT queries and each query is used by the server to compute multiple answers. It will be crucial for the security of the protocol that the client be unable to correlate answers with queries, beyond correlations which follow from the outputs. Such correlations will reveal to the client information about the structure of the server’s branching program.

Roughly speaking, our notion of strong OT strengthens the above server privacy requirement by requiring the distribution of the answer \( a \) conditioned on the output \( s_i \) to be independent of the query \( q \). In other words, the distribution of the answer depends on the output (and the public key) alone. It turns out that a natural implementation of one-round OT based on additively homomorphic encryption [94, 122] satisfies the required properties (see Section 5.4.1). We now formally define strong OT.
Definition 5.5. [Strong OT] A strong OT protocol is defined by a tuple of PPT algorithms \((G_{OT}, Q_{OT}, A_{OT}, D_{OT})\). The protocol involves two parties, a client and a server, where the server’s input is a \(t\)-tuple of strings \((s_1, \ldots, s_t)\) of length \(\tau\) each, and the client’s input is an index \(i \in [t]\). The parameters \(t, \tau\) are given as inputs to both parties. The protocol proceeds as follows:

- The client generates \((pk, sk) \leftarrow G_{OT}(1^k)\), computes a query \(q \leftarrow Q_{OT}(pk, 1^t, 1^\tau, i)\), and sends \((pk, q)\) to the server.
- The server computes \(a \leftarrow A_{OT}(pk, q, s_1, \ldots, s_t)\) and sends \(a\) to the client.
- The client computes and outputs \(D_{OT}(sk, a)\).

We require that if both parties follow the protocol, the client always outputs \(s_i\). We denote the length of the query \(q\) by \(\alpha(k, t, \tau)\) and the length of the answer \(a\) by \(\beta(k, t, \tau)\).

Our main protocol will require \(\beta(k, t, \tau) = \tau + \text{poly}(k, t)\) to efficiently accommodate BPs of arbitrary length. (In fact, it suffices that the above holds for \(t = 2\).) This will be our default efficiency requirement. However, this requirement can be relaxed if one settles for weaker forms of our main result that apply to shallow BPs, such as constant-length BPs over a polynomial-size input alphabet.

We now define the client privacy and (strong) server privacy requirements.

Definition 5.6. [Strong OT: client privacy] We require that the client’s query \(q\) keep \(i\) semantically secure. That is, the advantage of any PPT adversary \(\text{Adv}\) in the following game is negligible in the security parameter \(k\):

- \(\text{Adv}\) is given \(1^k\) and generates \(1^t, 1^\tau\) and \(i_0, i_1\) such that \(i_0, i_1 \in [t]\).
- Let \(b \leftarrow \{0, 1\}\), \((pk, sk) \leftarrow G_{OT}(1^k)\), and \(q \leftarrow Q_{OT}(pk, 1^t, 1^\tau, i_b)\).
- \(\text{Adv}\) is given the challenge \((pk, q)\) and outputs a guess \(b’\) for \(b\).

The advantage of \(\text{Adv}\) is defined as \(\text{Pr}[b = b’] - 1/2\).

Our strong variant of \(\text{Adv}\) is defined as \(\text{Pr}[b = b’] - 1/2\).

Definition 5.7. [Strong OT: server privacy] There exists an expected polynomial time simulator \(\text{Sim}_{OT}\) such that the following holds. For every \(k, t, \tau, i \in [t]\), pair \((pk, q)\) that can be generated by \(G_{OT}, Q_{OT}\) on inputs \(k, t, \tau, i\), and strings \(s_0, \ldots, s_{t-1} \in \{0, 1\}^\tau\), the distributions \(A_{OT}(pk, q, s_1, \ldots, s_t)\) and \(\text{Sim}_{OT}(pk, 1^t, s_i)\) are identical.

In the following it will sometimes be convenient to index the server’s inputs \(s_i\) by \(0, 1, \ldots, (t-1)\) instead of \(1, 2, \ldots, t\).
5.4 Main protocol

In this section we will describe our main protocol for evaluating branching programs on encrypted data. The protocol will provide client privacy, along with size hiding server privacy in the semi-honest model. Extensions that achieve server privacy in the malicious model will be presented in Section 5.5.

We fix a polynomially bounded length function $\ell(\cdot)$, and assume that if the client’s input $x$ is of length $n$, then the server’s BP $P$ is of length $\ell(n)$. (To conform to our general definition of representation models, one may define $P(x) = 0$ for $P$ and $x$ that do not match.) We also view the input domain $I$ and output domain $O$ as being implicitly determined by $n$. However, in the following it will be convenient to view $\ell$, $|I|$, and $|O|$ as separate parameters which are given to both parties, and analyze the complexity of the protocol as a function of these parameters. We will also assume that $P$ is layered (see Definition 2.6). As discussed in Section 2.4, every BP can be efficiently transformed into an equivalent layered BP without increasing its length.

Our protocol is based on a strong OT protocol as defined in Section 5.3.2 and proceeds roughly as follows. (For simplicity, assume that the input domain $I$ of $P$ is binary and that every nonterminal node in the graph has outdegree 2.) The client generates, for every input variable $x_i$ and level $j$, an OT query $q^j_i$ corresponding to a selection of the $x_i$-th string out of a pair of strings of an appropriate length. (This length will depend on $j$ and will be later understood from the context.) The $\ell n$ queries $q^j_i$ jointly form the encryption $c$ of $x$.

To evaluate $P$ on $c$, the server makes a bottom-up pass on $P$, starting with the terminal nodes $T$ and ending with the initial node $v_0$. This pass labels each node $v$ in the graph by an OT answer which encrypts the output value to which $x$ leads from this node. The pass consists of $\ell + 1$ iterations, where in iteration $j$ ($0 \leq j \leq \ell$) all nodes of height $j$ are handled. In iteration $0$ every terminal node $v$ is labeled by the corresponding output value $\psi_v(v)$. At the onset of the $j$-th iteration, $j \geq 1$, all nodes of height $j - 1$ have already been labeled. For each node $v$ of height $j$, we want the labeling of $v$ to encrypt the label of the child of $v$ to which $x$ leads. Such a label is computed by using the OT answering algorithm as follows. Suppose that the children of $v$ are $v_0$ and $v_1$, where $P$ branches from $v$ to $v_b$ if $x_i = b$. The label of $v$ then computed by applying the OT answering algorithm to the query $q^j_i$ on the pair of strings $(\text{label}(v_0), \text{label}(v_1))$. Note that since $P$ is layered, the two labels have the same length. Moreover, by the strong server privacy property of the OT protocol, the label of $v$ can be viewed as an encryption of the label of the selected child $v_{x_i}$. In particular, this label does not contain any information about the identity of the variable $x_i$ that was used to determine the selection. (If a standard one-round OT is used, this is not necessarily be the case.) Finally, at the end of iteration $\ell$, the initial node $v_0$ is labeled by an OT answer which can be viewed as an (iterated) encryption of the output value $P(x)$. The client decrypts $P(x)$ by applying the OT decryption algorithm $\ell$ times to the label of $v_0$.

The above protocol is formally described in Figure 5.1. Its correctness is implied by the following lemma, which can be easily proved by induction on the height $h$.

**Lemma 5.1.** For any node $v$, let $P_v(x)$ denote the output of $P$ on the input $x$ if $v$ is used as the initial node. Then, for every $0 \leq h \leq \ell$ and every node $v$ of height $h$ we have $D_{\text{OT}}^{(h)}(sk, \text{label}(v)) = P_v(x)$, where $D_{\text{OT}}^{(h)}(sk, \cdot)$ denotes the $h$-th iterate of $D_{\text{OT}}(sk, \cdot)$.

In particular, $D_{\text{OT}}^{(\ell)}(sk, \text{label}(v_0)) = P(x)$, from which correctness follows. We turn to analyze protocol’s efficiency.
Main Protocol

- Common inputs: security parameter $1^k$, a branching program length parameter $1^\ell$, input domain $I = \{0, 1, \ldots, t - 1\}$, output domain $O = \{0, 1\}$.
- Client input: an assignment $x = (x_1, \ldots, x_n) \in I^n$.
- Server input: a layered BP $P = (G(V, E), v_0, T, \psi_V, \psi_E)$ of length $\ell$.
- Sub-protocol: a strong OT protocol $(G_{OT}, Q_{OT}, A_{OT}, D_{OT})$ with answer length $\beta(k, t, \tau)$.

1. **Setup** $\text{Gen}(1^k)$:
   - Let $(pk, sk) \leftarrow G_{OT}(1^k)$.
   - Return $(pk, sk)$.

2. **Encryption** $\text{Enc}(pk, x)$:
   - For $1 \leq i \leq n$, generate a vector $q_i = (q_1^i, \ldots, q_\ell^i)$, where $q_j^i$ is obtained by:
     $$q_j^i \leftarrow Q_{OT}(pk, 1^\ell, 1^{\beta_j}, x_i),$$
   and where the lengths $\beta_j$ are defined by $\beta_1 = \log |O|$ and $\beta_{j+1} = \beta(k, t, \beta_j)$.
   - Return $c = (q_1, \ldots, q_n)$.

3. **Evaluation** $\text{Eval}(1^k, pk, c = (q_j^i), P)$:
   - Initialization: for each $v \in T$ set $\text{label}(v) \leftarrow \psi_V(v)$.
   - While $v_0$ is not labeled:
     - Pick an unlabeled node $v \in V - T$ such that all its children are labeled.
     - Let $i \leftarrow \psi_V(v)$ and $h \leftarrow \text{height}(v)$.
     - Let $\text{label}(v) \leftarrow A_{OT}(pk, q_h^i, \text{label}(u_0), \ldots, \text{label}(u_{t-1}))$, where $u_m$ is the (unique) node such that $m \in \psi_E(v, u_m)$.
     Note that the nodes $u_m$ are not necessarily distinct.
   - Return $c' = \text{label}(v_0)$.

4. **Decryption** $\text{Dec}(sk, c')$:
   - Let $d_\ell \leftarrow c'$.
   - For $j = \ell$ down to 1, let $d_{j-1} \leftarrow D_{OT}(sk, d_j)$.
   - Return $d_0$.

Figure 5.1: Evaluating a branching program on encrypted data
Efficiency. Recall that we denote the length of an OT query by $\alpha(k,t,\tau)$ and the length of an OT answer by $\beta(k,t,\tau)$. Let $\beta_j$ be as defined in Step 2, namely the result of applying the $j$-th iterate of $\beta(k,t,\cdot)$ on $\log|O|$. The length of the encryption $e$ computed by the client is then bounded by $\ell n \cdot \alpha(k,t,\beta_j)$ and the length of the ciphertext $e'$ computed by the server is $\beta_{\ell+1}$. By default, we assume the strong OT implementation to be such that $\beta(k,t,\tau) = \tau + \text{poly}(k,t)$. (See Section 5.4.1 for a concrete implementation using the Damgård-Jurik cryptosystem.) In such a case, the overall communication is $\text{poly}(k,n,\ell)$, which is in particular independent of $|P|$ as required. We will later present an optimized instantiation of the main protocol with a total communication of $O(kn\ell)$ (for the case of binary inputs and outputs). Finally, the computation performed by each party is polynomial in the length of its input.

Remark 5.4. When $\ell(n) \ll n$, the requirement that $\beta(k,t,\tau) = \tau + \text{poly}(k,t)$ can be relaxed. In particular, if $\ell(n) = O(\log n)$ it suffices that $\beta(k,t,\tau) = O(\tau) + \text{poly}(k,t)$. A strong OT protocol with the latter efficiency requirement can be based on additively homomorphic cryptosystems which expand the ciphertext length by a constant factor, such as El-Gamal (see Section 5.4.1). If $\ell(n) = O(1)$, we can rely on an arbitrary strong OT, which in turn can be based on an arbitrary additively homomorphic encryption scheme (including, for instance, the Goldwasser-Micali cryptosystem [65]).

Remark 5.5. The PIR protocol of Kushilevitz and Ostrovsky [94] can be viewed as an instance of our construction in which $\ell$ is set to some constant $d$, the input domain $I$ is of size $t = N^{1/d}$ (where $N$ is the database size), and the database is represented as a complete decision tree of depth $d$ and degree $N^{1/d}$. Its variant suggested in [122] (resp., [101]) corresponds to a decision tree of depth $\sqrt{\log N}$ and degree $t = 2^{\sqrt{\log N}}$ (resp., depth $\log N$ and degree $t = 2$). These three depth parameters correspond to the different BP length regimes discussed in Remark 5.4.

We turn to prove the security properties of the main protocol. In the following we assume that the given strong OT subprotocol is secure and that its answer complexity is $\beta(k,t,\tau) = \tau + \text{poly}(k,t)$. In Section 5.4.1 we will show that this assumption is implied by the DCRA assumption.

Theorem 5.1. The protocol described in Figure 5.1 provides client privacy according to Definition 5.2 as well as perfect size hiding server privacy in the semi-honest model according to Definition 5.4.

Proof. Client privacy readily follows from the client privacy requirement in the underlying OT protocol. The security of sending polynomially many strong OT queries under the same key follows from the security of encrypting multiple messages under the same key in public-key encryption schemes (see [60], Theorem 5.2.11).

To prove size hiding server privacy, we describe a perfect simulator $\text{Sim}$. The idea is to recreate the labels of the computation path from $v_0$ to a terminal node labeled with $P(x)$ without knowing the nodes traversed by the path. $\text{Sim}$ will use the OT simulator $\text{Sim}_{\text{OT}}$ as a subroutine. On inputs $(1^k,1^{\ell_x},pk,P(x))$ (and given $|I| = t$ as an additional public input), $\text{Sim}$ proceeds as follows:

- Let $\ell \leftarrow \ell(|x|)$, $\lambda_0 \leftarrow P(x)$.
- For $j = 1$ to $\ell$, let $\lambda_j \leftarrow \text{Sim}_{\text{OT}}(pk,1^t,\lambda_{j-1})$.
- Return $\lambda_\ell$.

111
5.4.1 Implementing strong OT

Our concrete implementation of strong OT is based on the Damgård-Jurik (DJ) additively homomorphic public-key cryptosystem [43], which generalizes Paillier’s cryptosystem [111]. It is suitable for our needs because it allows us to encrypt a group element of length \( \tau \) into a ciphertext of length \( \tau + O(k) \), where \( k \) is a security parameter. This efficiency feature is unique among all known additively homomorphic encryption schemes and is needed for our main protocol to be efficient for arbitrary length bounds \( \ell \). The semantic security of the DJ cryptosystem follows from the Decisional Composite Residuosity Assumption (DCRA) [43].

We now describe the main properties of the DJ cryptosystem that are useful for our purposes (see [43] for further details).

- **KEY GENERATION**: Given a security parameter \( k \), \( \text{Gen}(1^k) \) outputs a secret key \((p_1, p_2)\), where \( p_1, p_2 \) are random \( k \)-bit primes (i.e., \( 2^{k-1} \leq p_1, p_2 < 2^k \)), and a public key \( N = p_1 p_2 \). The above choice of \( p_1, p_2 \) guarantees that \( \gcd(N, \phi(N)) = 1 \). This property will be useful in what follows. We refer to \( N \) which can be generated by \( \text{Gen}(1^k) \) as a valid DJ key.

- **ENCRYPTION**: The DJ cryptosystem is length-flexible in the sense that every fixed key \( N \) allows to encrypt plaintexts of an arbitrary (polynomial) length, where the encryption only adds \( O(k) \) bits to the length of the plaintext. Given a plaintext length parameter \( e \), where \( 1 \leq e < \min(p_1, p_2) \), we define a plaintext group \( M_{N,e} = \mathbb{Z}_{N^e} \) and a ciphertext group \( C_{N,e} = \mathbb{Z}_{N^e+1}^* \). The restriction on \( e \) is required for correct decryption, and since we will only use \( e \leq \text{poly}(k) \) it will always hold. Now fix some valid pair \((N, e)\). To abbreviate notation we denote the ciphertext group \( C_{N,e} = \mathbb{Z}_{N^e+1}^* \) by \( C \). Let \( C_0 = C^{N^e} = \{c^{N^e} \mid c \in C\} \). Clearly, \( C_0 \) is a subgroup of \( C \). Let \( g = N + 1 \in C \). The output distribution of the encryption is specified via an injective homomorphism \( H : M_{N,e} \rightarrow C/C_0 \) defined by \( H(m) = g^m \cdot C_0 \), where \( g^m \cdot C_0 \) denotes the coset represented by \( g^m \) in \( C/C_0 \). To encrypt \( m \in M_{N,e} \), the encryption function \( E_{N,e}(m) \) returns a random element in the coset \( H(m) \). This can be done by sampling \( r \leftarrow \mathbb{Z}_{N^e+1}^* \) and outputting \( c = g^m \cdot r^{N^e} \), where all multiplications are in \( C \).\(^5\) In particular, an encryption of \( 0 \) is a random element of \( C_0 \). Note that the difference between the size of the ciphertext \( \lceil \log(N^{e+1}) \rceil \) and the size of the plaintext \( \lceil \log(N^e) \rceil \) is indeed only \( O(k) \).

- **DECRIPTION**: Given \( c = g^m \cdot r^{N^e} \) and the factorization \((p_1, p_2)\) of \( N \), it is possible to efficiently decrypt \( m \). We denote the decryption algorithm by \( D_{(p_1, p_2), e}(c) \).

- **HOMOMORPHISM**: Given two ciphertexts \( c \in E_{N,e}(m) \) and \( c' \in E_{N,e}(m') \), their product \( c \cdot c' \) (in the ciphertext group) is a valid encryption of the sum \( m + m' \) (in the plaintext group). It follows that \( c^0 \)

\(^5\)In the original DJ cryptosystem, \( r \) is sampled from \( \mathbb{Z}_N^* \). The modification here is introduced in order to simplify the analysis of the construction for the malicious setting.
is an encryption of $\rho \cdot m$. Moreover, multiplying $c$ by a random encryption of 0 rerandomizes $c$ into a random encryption of $m$.

**Strong OT from the DJ cryptosystem.** The following strong OT protocol is similar to the PIR protocol of [94] and its generalizations from [122, 101]. The choice of DJ as the underlying cryptosystem is motivated by the goal of handling branching programs of an arbitrary length. If the length function $\ell(n)$ is small, other additively homomorphic cryptosystems can be used (see Remark 5.4).

**Construction 5.1.** [Strong OT] Let $(\text{Gen}, E_{N,e}, D_{(p_1,p_2),e})$ be the DJ cryptosystem. The OT protocol $(G_{OT}, A_{OT}, Q_{OT}, D_{OT})$ proceeds as follows.

1. $G_{OT}(1^k)$:
   - Let $(N, (p_1, p_2)) \leftarrow \text{Gen}(1^k)$.
   - Return $(N, (p_1, p_2))$.

2. $Q_{OT}(N, 1^k, 1^t, 1^\tau, i)$:
   - Let $e$ be the minimal integer such that $N^e > 2^\tau$. We naturally identify strings in $\{0, 1\}^\tau$ with integers in $M_{N,e} = \mathbb{Z}_{N^e}$, and assume that elements in the groups $M_{N,e}$ and $C_{N,e}$ are padded so that their representation reveals $e$.
   - Let $q_i \leftarrow E_{N,e}(1)$ and $q_j \leftarrow E_{N,e}(0)$ for all $j \in [t] \setminus i$.
   - Return $q = (q_1, \ldots, q_{t-1})$.

3. $A_{OT}(N, q, s_1, \ldots, s_t)$:
   - Infer $e$ from $q$.
   - Let $q_i \leftarrow E_{N,e}(1) \cdot (\prod_{i=1}^{t-1} q_i^{s_i})^{-1}$ (where all operations are in $C_{N,e}$).
   - Let $a \leftarrow \prod_{i=1}^t q_i^{s_i} \cdot E_{N,e}(0)$.
   - Return $a$.

4. $D_{OT}((p_1, p_2), a)$:
   - Infer $e$ from $a$.
   - Return $D_{(p_1,p_2),e}(a)$.

**Analysis.** Correctness follows by observing that $(q_1, \ldots, q_t)$ encrypt the $i$-th unit vector of length $t$ and $a$ encrypts the inner product of $(s_1, \ldots, s_t)$ with this vector, which yields $s_i$. Client privacy follows from the semantic security of the DJ cryptosystem, which can be based on the DCRA assumption [43]. Server privacy follows from the fact that (due to rerandomization) the server’s answer on any valid $q$ is a random encryption of $s_i$, which can be easily generated by $\text{Sim}_{OT}$. The protocol’s query length is $\alpha(k, t, \tau) = t \cdot (\tau + O(k))$ and its answer length is $\beta(k, t, \tau) = \tau + O(k)$. 

113
5.4.2 Optimizations

**Optimizing the server’s work.** Our main protocol requires the branching program $P$ to be layered. Converting an arbitrary BP to an equivalent layered BP of the same length may generally result in a quadratic blowup to its size, which in turn results in a quadratic computational overhead on the server’s part. (We note, however, that most “natural” BPs, including ones that arise from other computation models such as finite automata, are either already layered or can be turned into equivalent layered BPs with only a linear overhead.) The quadratic overhead can be easily avoided in general if only client privacy is required. The main protocol can be modified in this case to operate on a non-layered BP by padding the labels that serve as OT inputs to match the size of the longest label.

**Optimizing the encryption length.** In the main protocol, the length of the encryption $c$ produced by $\text{Enc}$ must be bigger than $\sum_{j=1}^{\ell} \beta_j > \ell^2$. It turns out that the quadratic dependence on $\ell$ can be avoided by exploiting the specific structure of the DJ cryptosystem. The improvement is based on the following observation:

**Observation 5.1.** For every valid DJ key pair $(N, e)$, $e' < e$, $m \in M_{N,e}$ and $c \in E_{N,e}(m)$ (i.e., $c$ is some valid encryption of $m$) it holds that

$$c \mod N^{e'+1} \in E_{N,e'}(m \mod N^{e'}).$$

It follows from Observation 5.1 that the ciphertext $c$ may consist of $n$ encryptions $q_i$ in the largest group (rather than $n$ encryptions $q_j^i$ for every level $j$ of the BP), since the server can convert encryptions from the largest group into encryptions from smaller groups. (Note that since we only encrypt 0’s and 1’s, the conversion does not modify the encrypted value.) The improved implementation achieves communication complexity of $O(kn\ell)$ bits from the client to the server (instead of $O(kn\ell^2)$ in the original implementation) and $O(k\ell)$ bits from the server to the client (as in the original implementation). Clearly, the optimization does not compromise client or server privacy. Thus, we have:

**Theorem 5.2.** Assuming DCRA [43], there is a protocol for evaluating a binary branching program of length $\ell$ and of arbitrary size on an encrypted input of length $n$, with a total communication of $O(kn\ell)$ bits (where $k$ is a security parameter). The protocol provides client privacy, as well as size hiding server privacy in the semi-honest model.

**Remark 5.6 A generalization.** In the standard definition of branching programs, terminal nodes are labeled by fixed output values. Our main protocol can be generalized in a straightforward way to evaluate a more general class of branching programs, whose terminal nodes are labeled by functions of the input which can be efficiently evaluated on encrypted data. (The output of such a branching program is simply the output of the function labeling the terminal node reached by the computation path.) For instance, using the BGN cryptosystem [21] the main protocol can be generalized, with no asymptotic efficiency overhead, to evaluate branching programs whose terminal nodes are labeled by degree-2 polynomials or 2-DNF formulas in the inputs.
5.5 Handling malicious clients

In this section we describe the required modifications for achieving security against malicious clients. For simplicity, we restrict the attention throughout this section to the case of branching programs over binary inputs.

We start by observing that a malicious client can easily break the server privacy of the main protocol even if it honestly generates the public key $pk$.

**Example 5.1.** Consider a client who sends an encryption of 2 (instead of 0 or 1) as an OT query. In this OT invocation, the client can recover both $s_0$ and $s_1$. This potentially reveals additional information about the structure of the branching program $P$. For instance, in the degenerate case where $P$ consists of an initial node and two terminal nodes, the client will learn the values of both terminal nodes.

The above mild form of cheating is relatively easy to handle using previous techniques [59, 1, 96] and will be addressed in Section 5.5.1. A more challenging goal is to handle clients that are also free to choose invalid public keys $pk$. This scenario will be addressed in Section 5.5.2.

Before describing our solutions, we formalize our notions of server privacy in the malicious model. The following definitions modify Definition 5.4 in that they allow an unbounded simulator to extract an effective input $x^*$ from a corrupted ciphertext $c^*$ and a (possibly) corrupted public key $pk^*$. The use of unbounded simulation seems necessary in the “vanilla” one-round malicious model (i.e., without setup assumptions) and was previously made in similar contexts [106, 1, 54, 82, 96]. On the other hand, the type of server privacy we realize is stronger than the traditional one in that it holds with respect to computationally unbounded clients. Thus, we get a pure form of “information-theoretic” server privacy.

We start by defining the trusted setup model, where the client is forced to use a valid public key $pk$ but can cheat by creating invalid ciphertexts $c^*$. This model is motivated by the fact that the same public key may be reused to encrypt many different inputs. Thus, one can afford an expensive certification procedure (e.g., using interactive zero-knowledge proofs or a trusted party) that is used once and for all.

**Definition 5.8.** [Size hiding server privacy: trusted setup model] Let $\Pi = (Gen, Enc, Eval, Dec)$ be a protocol for evaluating programs from a representation model $U$ on encrypted data. We say that $\Pi$ has statistical size hiding server privacy in the trusted setup model if there exists a computationally unbounded, randomized algorithm $Sim$ and a negligible function $\epsilon(\cdot)$ such that the following holds. For every security parameter $k$, valid public key $pk$ that can be generated by $Gen(1^k)$, and arbitrary ciphertext $c^*$ there exists an “effective” input $x^*$ such that for every program $P \in \{0, 1\}^*$, we have

$$SD(Eval(1^k, pk, c, P), Sim(1^k, pk, c^*, U(P, x^*))) \leq \epsilon(k).$$

The case of computational server privacy is defined in an analogous way (see Definition 5.3), where statistical indistinguishability is replaced by computational one.

We turn to the fully malicious model.

**Definition 5.9.** [Size hiding server privacy: fully malicious model] We say that $\Pi$ has (statistical or computational) size hiding server privacy in the fully malicious model if it satisfies Definition 5.8 with the following
modification: instead of quantifying over all valid public keys $pk$, now the quantification is over arbitrary public keys $pk^*$.

Protocols for computing on encrypted data in the above model give rise to one-round (two-message) protocols for secure two-party computation of $U(\cdot, \cdot)$ under a relaxed security definition as in [106, 1, 54].

A natural approach for handling malicious clients would be to leave the main protocol as it is and only upgrade the original strong OT primitive into one that achieves security against malicious clients. Unfortunately, we cannot use this modular approach for several reasons. First, the basic variant of the protocol requires the client to use each input $x_i$ in multiple OT invocations (corresponding to the different levels where $x_i$ appears) and so the client could cheat by simply using inconsistent inputs in these OT invocations. Second, it is not straightforward to construct a strong OT protocol which simultaneously satisfies both our security and efficiency requirements in the malicious model. It is interesting to note that a one-round OT protocol of Kalai [82], which is based on Paillier’s cryptosystem and can be generalized to work with the DJ cryptosystem, fails with respect to both security (in that it is not a strong OT) and efficiency (in that its answer significantly blows up the length of the selected string). Still, ideas from [82] will be very instrumental in our solution for the fully malicious model.

5.5.1 Trusted setup model

We now describe a solution in the trusted setup model. Our starting point is the optimized instantiation of the protocol for the semi-honest model (Section 5.4.2), where in the case of binary inputs ($t = 2$) the client sends a single encryption for each input. Our goal is to prevent the type of attack described in Example 5.1, namely to ensure that each encryption sent by the client is indeed an encryption of 0 or 1. To this end one could employ general-purpose zero-knowledge proofs, forcing the client to prove that its queries are well formed. However, this approach requires either multiple rounds of interaction or setup assumptions which we would like to avoid, and also involves a considerable efficiency overhead.

Instead, we apply the conditional disclosure of secrets (CDS) methodology of [59, 1]. The idea is that instead of making the client prove that its queries are well formed, it suffices for the server to disclose its answer $c'$ to the client only under the condition that the queries are well formed. Using the homomorphic property of the encryptions, the latter conditional disclosure can be done without the server even knowing whether the condition is satisfied.

The original CDS solutions from [1] relies on additively homomorphic encryption over groups of a prime order. An efficient extension to groups of a composite order was suggested in [96], assuming that the order of the group is sufficiently “rough”. We employ a similar extension which avoids the roughness assumption and is geared towards the solution in the fully malicious model.

We start by describing the approach of [1]. The simplest setting involves a server holding a (valid) public key $pk$ of an additively homomorphic cryptosystem, a ciphertext $c \in E_{pk}(m)$ (presumably generated by a client), and a secret $s$. The client holds the secret key $sk$ corresponding to $pk$. The goal is for the server to send a single (randomized) ciphertext $\mu$ such that: (1) if $m = 0$ then $s$ can be recovered from $\mu$ using the secret key; and (2) if $m \neq 0$ then $\mu$ reveals (almost) no information about $s$. The above is referred to as a CDS of the secret $s$ under the condition $m = 0$. As in [59, 1], a solution to this simple CDS problem can be easily extended to CDS under more general conditions, involving multiple inputs $m_i$ and general predicates over atomic conditions of the form $m_i = b_i$. In particular: (1) to disclose $s$ under an atomic condition
By dividing $m_i$ by $b_i$, where $b_i \neq 0$, we divide $c$ by an encryption of $b_i$ and then apply CDS under the condition $m_i = 0$; (2) to disclose $s$ under a disjunction of conditions $C_1 \lor C_2$, we independently disclose $s$ under $C_1$ and under $C_2$; and (3) to disclose $s$ under a conjunction $C_1 \land C_2 \land \cdots \land C_n$, we pick $n$ random shares $s_1, \ldots, s_n$, disclose each $s_i$ under the corresponding condition $C_i$, and output also $s' = s \oplus s_1 \oplus s_2 \oplus \cdots \oplus s_n$. This implies that $2n$ invocations of the above atomic CDS primitive are sufficient to disclose a secret under the suitable condition here, namely that $n$ ciphertexts $c_i$ all encrypt $0/1$ values.

The implementation from [1] of atomic CDS as above (under the condition $m = 0$) lets $\mu$ be a random encryption of $s + \rho m$, where $\rho$ is a random integer between $1$ and the order of the plaintext group. Note that $\mu$ can be efficiently computed using the homomorphic properties of the encryption. Requirement (1) holds because if $m = 0$ then $\mu$ encrypts $s + \rho \cdot 0 = s$. Requirement (2) holds in the case where the plaintext group is of a prime order; indeed, in this case if $m \neq 0$ then $\rho m$ is uniformly distributed over the plaintext group and can thus be used to hide $s$.

The next observation is that even in the case that the plaintext group has a composite order, not all is lost. In this case, if $m \neq 0$ then $\rho m$ is uniformly distributed over a nontrivial subgroup of the plaintext group. If $s$ is chosen uniformly at random from the plaintext group, then $s$ will still have at least one bit of remaining entropy even when conditioned on $\mu$. This residual randomness can be extracted using standard privacy amplification techniques. Specifically, to disclose an $l$-bit secret we first repeat the above $l + k$ times with independent secrets $s_i$, increasing the conditional entropy to $l + k$, and then apply an arbitrary strong randomness extractor [108] (e.g., a pairwise independent hash function[17, 71, 16]) to extract $l$ (almost) perfectly secret bits from the partially leaked secrets. More explicitly, Following is a statistical version of the CDS, conditioning on $\bigwedge_i(q_i \in E_{N,e}(0) \lor q_i \in E_{N,e}(1))$, as we need in our protocol.

- Assume we need to disclose an $l$-bit secret. Pick an $l$-bit value independently at random.
- For the “atomic” conditions, of the form “$q_i$ encrypts $b$”, involved, prepare a random secret $s_{i,b} \in \{0,1\}^l$ to disclose, and $l + k$ random values $s_{1,i,b}, \ldots, s_{l+k,i,b}, \rho_{l+k}$ in $\mathbb{Z}_{N^e}$. Pick randomness $r$ for the extractor $\text{Ext}$, and send

\[
(s_{i,b} \oplus \text{Ext}(s_{1,i,b}, \ldots, s_{l+k,i,b}, r), E_{N,e}(s_{1,i,b}) (E_{N,e}(b) q_i^{-1})^\rho_1, \ldots, E_{N,e}(s_{l+k,i,b}) (E_{N,e}(b) q_i^{-1})^\rho_{l+k}).
\]

- At the highest level, we disclose $s$ as the secret, by combining outputs of CDS executions as explained above to obtain a CDS under the complex condition $\bigwedge_i(q_i \in E_{N,e}(0) \lor q_i \in E_{N,e}(1))$.

We modify the protocol in Figure 5.1 so that it remains server-private in the trusted setup setting. We start with an implementation where Strong-OT is instantiated via DJ as in Construction 5.1. We also utilize the optimization where the queries only encrypt values of the $x_i$’s for the highest level (see Section 5.4.2). The required modification is simply:

- Instead of sending $d_0$ in the clear, as done in Figure 5.1, send $d_0 \oplus s$, where $s$ is the secret generated and revealed by the CDS as above (or a computational version of it, if settling for computational server privacy), which conditions on the encryptions $q_i$ received are each an encryption of either 0 or 1.
- Additionally, send the CDS output $\mu$.
- The client first recovers $d_0$ from $\mu$, $d_0 \oplus s$, and proceeds as before on $d_0$, to recover the output.
The above approach (or the similar approach from [96]) solves our problem in the trusted setup model. In this case, every possible string \( c^* \) can be interpreted as a valid ciphertext encrypting some message \( m \) in the plaintext group \( \mathbb{Z}_{N^e} \). (If \( c^* \) is not in the ciphertext group \( \mathbb{Z}_{N^e+1}^* \), which can be efficiently recognized, we let \( c^* = 1 \).) Thus, we can use the above to disclose the server’s answer under the condition that the \( n \) encryptions produced by the client are well formed. This yields a protocol for the trusted setup model with the same asymptotic communication complexity as the optimized version of the original protocol.\(^6\)

5.5.2 Fully malicious model

The solution in the trusted setup model relied on the fact that for a valid public key \( N \), the DJ cryptosystem is additively homomorphic and valid ciphertexts in \( \text{support}(E_{N,e}(\mathbb{Z}_{N^e})) \) can be efficiently recognized.

When \( N \) is malformed, \( \text{support}(E_{N,e}(\mathbb{Z}_{N^e})) \) may be a proper subgroup of \( \mathbb{Z}_{N^e+1}^* \), and it is not clear how to efficiently restrict ciphertexts \( c^* \) provided by a malicious client to be taken from this subgroup\(^7\). In fact, we will demonstrate an explicit attack against the protocol in Section 5.5.1 in which the client sends a malformed key \( N \), along with a carefully selected malformed ciphertext \( c^* \in \mathbb{Z}_{N^e+1}^* \setminus \text{support}(E_{N,e}(\mathbb{Z}_{N^e})) \), and learns extra information about the branching program (beyond what is allowed by Definition 5.9). Such “harmful” pairs \( N, c^* \) are not known to be efficiently recognizable. Our general approach for overcoming this limitation is to efficiently transform an arbitrary ciphertext \( c \) into a different ciphertext \( c' \) such that \( c' \) maintains the “useful information” in \( c \) when \( c \) is well formed, and yet is guaranteed to be harmless even when \( c \) is malformed.

We start with some useful notation and technical lemmas about the structure of \( \mathbb{Z}_{N^e+1}^* \).

**Notation.** We denote by \( \text{ord}_G(v) \) the order of an element \( v \) in a group \( G \). When \( G \) is omitted it is either the default group \( \mathbb{Z}_{N^e+1}^* \) or a different group which is understood from the context. For a direct product group \( G = G_1 \times \ldots \times G_t \), we denote elements of \( G \) by \( (v_1, \ldots, v_t) \), as usual. We denote by \( (H_1, \ldots, H_t) \), where each \( H_i \) is a subset of \( G_i \), the subset \( H_1 \times \ldots \times H_t \subseteq G \). In this notation, we may substitute a subset \( H_i \) by either the symbol ‘\( * \)’, indicating that \( H_i = G_i \), or by some fixed element \( v_i \in G_i \), indicating that \( H_i = \{v_i\} \).

For instance, \( (*, *, 0) \) denotes the subgroup \( G_1 \times G_2 \times \{0\} \). Finally, we let \( [N] \) denote the set of integers \( \{0, \ldots, N-1\} \) and let \( \phi(N) \) denote Euler’s totient function (that is, \( \phi(N) = |\mathbb{Z}_N^*| \)). We denote the “min-entropy” of a random variable \( X \) with a finite support, by \( H_\infty(X) = \min_{i \in \text{support}(X)} (- \log Pr(X = i)) \). For an event \( A \), \( H_\infty(X|A) \), denotes the infinity norm of \( X \), conditioned that event \( A \) occurred. We denote \( E_{N,e}(m) = g^m \), that is, the encryption of \( m \) where \( r \) is fixed to 1.

In the following we prove useful facts about the structure of \( \mathbb{Z}_{N^e+1}^* \) for arbitrary odd \( N \) and positive integers \( e \). We start by showing that the element \( g = N + 1 \) generates a subgroup of size \( N^e \) even when \( N \) is not a valid DJ public key. For this we rely on the following technical lemma.

\(^6\)This holds for the case of computational server privacy, where we can afford to disclose a short secret \( s \) and then encrypt the (long) answer using this key. The statistically private variant involves an additional multiplicative overhead of \( O(\ell) \), as evident from the sketch above.

\(^7\)We extend \( E_{N,e} : \mathbb{Z}_{N^e} \to \mathbb{Z}_{N^e+1}^* \) to denote the output of the encryption procedure of DJ to the case of possibly malformed keys in the natural way. That is, we let \( E_{N,e}(m) = (N + 1)^m \cdot N^e \mod (N^{e+1}) \), where \( r \) is uniform in \( \mathbb{Z}_{N^{e+1}}^* \), without concerning ourselves with the question whether the resulting encryption scheme remains useful and secure.
Lemma 5.2. Let \( p \) be an odd prime and \( N = q \cdot p^d \) where \( q \) is coprime to \( p \) and \( d \) is a positive integer. Let \( g_p = (N + 1) \mod p^h \), where \( h > d \). Then \( \text{ord}_{Z_{p^h}^*} (g_p) = p^{h-d} \).

Proof. Given \( 0 \leq x \leq h \), let \( G_{p,h,x} \) denote the subgroup \( \{ ip^x + 1 | 0 \leq i < p^{h-x} \} \) of \( Z_{p^h}^* \). Since \( Z_{p^h}^* \) is cyclic (by Fact 5.2), any \( G_{p,h,x} \) must also be cyclic. Clearly, \([G_{p,h,x}] = p^{h-x} \). The number of generators of \( G_{p,h,x} \) equals \( \phi([G_{p,h,x}]) = \phi(p^{h-x}) = p^{h-x-1}(p-1) \). All elements in \( G_{p,h,d+1} \) cannot generate \( G_{p,h,d} \) since their order divides \( p^{h-d-1} \). Therefore, by counting, all elements in \( G_{p,h,d} \setminus G_{p,h,d+1} \) (there are \( p^{h-d-1}(p-1) \) such elements) must be generators of \( G_{p,h,d} \). These are precisely elements of the form \( q p^d + 1 \), where \( q \) is coprime to \( p \). Note that \( g_p \) is of this form. (Indeed, \( g_p = (q \cdot p^d + 1) \mod p^h = (q' \cdot p^{d+1}) \mod p^h \) for some \( q' < p^{h-d} \) where \( q' \equiv q \mod p \); hence if \( q \) is coprime to \( p \) then so is \( q' \).) We conclude that \( \text{ord}(g_p) = p^{h-d} \) as required. \( \square \)

Claim 5.3. [Order of \( N + 1 \) in \( Z_{N+1}^* \)] Let \( N \) be an odd integer and \( e \) a positive integer. Then

\[
\text{ord}_{Z_{N+1}^*} (N + 1) = N^e.
\]

Proof. Let \( N = \prod_{i=1}^t p_i^{d_i} \) be the (unique) decomposition of \( N \) into powers of distinct primes, let \( P_i = p_i^{d_i+1} \), \( G_i = Z_{P_i}^* \), and \( g_i = (N+1) \mod P_i \). It follows from Fact 5.1 (2) that \( Z_{N+1}^* \) is isomorphic to \( G_1 \times \ldots G_t \) via the isomorphism \( \psi(x) = (x \mod P_1, \ldots, x \mod P_t) \) and thus \( \text{ord}_{Z_{N+1}^*} (N + 1) = \text{lcm} (\text{ord}_{G_i}(g_i)) \). Applying Lemma 5.2 for each \( i \) with \( p = p_i, \) \( d = d_i \) and \( h = d_i(e + 1) \), we conclude that \( \text{ord}_{G_i} (g_i) = p_i^{d_i(e+1)-d_i} = p_i^{ed_i} \). Finally, since the \( p_i \) are distinct primes, we have \( \text{lcm}(p_1^{ed_1}, \ldots, p_t^{ed_t}) = \prod_{i=1}^t p_i^{ed_i} = N^e \) as required. \( \square \)

The characterization given in the following lemma is central in our analysis. It states that for odd \( N \), the group \( Z_{N+1}^* \) is isomorphic to a direct product of the form \( P \times C_0 \times C_1 \), where \( P, C_0, C_1 \) are Abelian groups with properties that will be specified in the lemma. In the following we will use additive notation for the groups \( P, C_0, C_1 \). For instance, for an integer \( a \) and a group \( H \) we let \( aH \) denote the group \( \{ ah | h \in H \} \), and denote by \( h + H' \) a coset of \( H' \) in \( H \) (where \( h \in H \) and \( H' \) is a subgroup of \( H \)). When considering elements of \( Z_{N+1}^* \) directly, rather than as elements of \( P \times C_0 \times C_1 \), we still use multiplicative notation, similarly to the notation used when defining the DJ cryptosystem.

Lemma 5.3. [Decomposition of \( Z_{N+1}^* \)] Let \( N > 1 \) be an odd integer and \( e \) a positive integer. Let \( N = \prod_{i=1}^t p_i^{d_i} \) be the (unique) decomposition of \( N \) into powers of distinct primes. Let \( S = \bigtriangleup \prod_{i=1}^t p_i \) and let \( P \triangleleft Z_{N+1}/S \). Then the group \( Z_{N+1}^* \) is isomorphic to a group \( F = P \times C_0 \times C_1 \), where \( C_0 \) is of size coprime to \( N \), \( C_1 \) is of size whose prime divisors all divide \( N \), and \( |C_0 \times C_1| = \phi(S) \). Furthermore, there exists an isomorphism from \( Z_{N+1}^* \) to \( F \) which maps \( g = N + 1 \) to \((\tilde{g}, 0, 0)\) for some \( \tilde{g} \in P \setminus \{0\} \).

Proof. We proceed by constructing an isomorphism between \( Z_{N+1}^* \) and \( F \) as required. Let \( P_i = p_i^{d_i+1} \), \( G_i = Z_{P_i}^* \), \( g_i = (N+1) \mod P_i \), and \( F' = G_1 \times \cdots \times G_t \). It follows from Fact 5.1 (2) that \( Z_{N+1}^* \) is isomorphic to \( F' \). Using Fact 5.2, \( F' \) is isomorphic to \( F'' = Z_{P_1} \times Z_{P_1} \times \ldots \times Z_{P_t} \). Consider an isomorphism \( \psi'' \) between \( Z_{N+1}^* \) and \( F'' \), obtained by composing the isomorphism \( \eta \) from Fact 5.1 and any isomorphism between \( F' \) and \( F'' \) as above (that is, one that maps \((Z_{P_1}^*, 0, \ldots, 0) \rightarrow (Z_{P_1}^*, Z_{P_1}, 0, 0, \ldots, 0, 0) \), etc.). At this point, \( g = N + 1 \) is mapped to an element in \((*, 0, \ldots, *, 0) \):
otherwise, for some \(i\), \(\text{ord}_{\mathbb{Z}_{P_i}^*}(g_i)\) would have a nontrivial factor which is coprime to \(p_i\), in contradiction to Lemma 5.2. Finally, \(F''\) is mapped onto \(F\) via the following isomorphism.

- Rearrange the factors in \(F''\) to obtain \((\mathbb{Z}_{P_1/p_1} \times \ldots \times \mathbb{Z}_{P_i/p_i}) \times \mathbb{Z}_{p_{i+1} - 1} \times \ldots \times \mathbb{Z}_{p_{t} - 1}\), which using Fact 5.1 (1) is isomorphic to \(P \times \mathbb{Z}_{p_{i+1} - 1} \times \ldots \times \mathbb{Z}_{p_{t} - 1}\).

- For each \(i\), let \(p_i - 1 = \prod_{j=1}^{t_i} p_{i,j}^{d_{i,j}}\) be the (unique) decomposition of \(p_i - 1\) into powers of distinct primes. Repeatedly applying Fact 5.1 (1), we get that \(\mathbb{Z}_{p_{i+1} - 1}\) is isomorphic to \(H_i = \mathbb{Z}_{p_{i+1} - 1} \times \ldots \times \mathbb{Z}_{p_{i} - 1}\), Replace \(\mathbb{Z}_{p_{i+1} - 1}\) with \(H_i\). Note that the order of each factor of \(H_i\) is a prime power.

- Regroup the factors of the groups \(H_i\) to \(C_0, C_1\) as required in the lemma, by moving factors of size \(p^d\) for some \(p\) coprime to \(N\) to the end (that is, every such factor appears after all factors whose size is coprime to \(N\)). We have obtained a group \(P \times C_0 \times C_1\) with properties as required.

Note that the above procedure can be formally described in terms of composing isomorphisms that follow the decompositions and regroupings. Finally, any isomorphism from \(\mathbb{Z}_{N_{e+1}}^*\) to \(P \times C_0 \times C_1\) obtained in this manner maps \(N + 1\) to \((*, 0, 0)\) (since for any group homomorphism, the zero element is mapped to the zero element).

In what follows we associate with any odd integer \(N\) and positive integer \(e\) an isomorphism \(\psi : \mathbb{Z}_{N_{e+1}}^* \rightarrow F\), where \(F\) and \(\psi\) are as in Lemma 5.3, and identify between elements of \(\mathbb{Z}_{N_{e+1}}^*\) and triples from \(F\) via this isomorphism. We illustrate the lemma by examining the structure of \(\mathbb{Z}_{N_{e+1}}^*\) for some useful cases.

**Example 5.2.** Consider a valid DJ key \(N = p_1p_2\), where \(p_1, p_2\) are \(k\)-bit primes (this implies in particular that \(\text{gcd}(\phi(N), N) = 1\)), and let \(e\) be a positive integer. Applying the decomposition procedure in the proof of Lemma 5.3, \(\mathbb{Z}_{N_{e+1}}^*\) is isomorphic to \(F = \mathbb{Z}_{N^e} \times \mathbb{Z}_{p_1 - 1} \times \mathbb{Z}_{p_2 - 1}\). (Unlike the proof of the lemma, here we do not fully decompose \(C_0\)) Indeed, \(S = p_1p_2 = N\), implying that \(P = \mathbb{Z}_{N^e}\), and \(|C_0 \times C_1| = \phi(S) = \phi(N)\). Also, since, \(\phi(N) = (p_1 - 1)(p_2 - 1)\) is coprime to \(N\), \(|C_0| = \phi(N)\), and \(C_1\) must be the trivial group \(\{0\}\). This is in agreement with what we know about the structure of \(\mathbb{Z}_{N_{e+1}}^*\) for well formed DJ keys \(N\).

We next see that the group \(C_1\) may be non-trivial when \(N\) is not a valid DJ key.

**Example 5.3.** Let \(p_1\) be an odd prime such that \(p_2 = 2p_1 + 1\) is prime, let \(N = p_1p_2\), and let \(e = 1\). In this case, \(\mathbb{Z}_{N_{e+1}}^*\) is isomorphic to \(F = \mathbb{Z}_N \times \mathbb{Z}_2 \times \mathbb{Z}_{p_1 - 1} \times \mathbb{Z}_{p_2 - 1}\). Indeed, \(\phi(S) = (p_1 - 1)(p_2 - 1) = 2p_1(p_1 - 1)\) and \(\text{gcd}(p_1 - 1, N) = 1\).

We will now see that \(E_{N,e}(Z_{N,e})\) is distributed over a proper subgroup of \(\mathbb{Z}_{N_{e+1}}^*\) whenever \(C_1\) is non-trivial. More generally, we use Lemma 5.3 to analyze the distribution of \(E_{N,e}(m)\) for an arbitrary odd \(N\) and \(m \in \mathbb{Z}_{N^e}\). Recall that the DJ encryption function \(E_{N,e}(m)\) samples a random element \(r\) from \(\mathbb{Z}_{N_{e+1}}^*\) and outputs \(g^{m_rN^e}\) (where \(g = N + 1\) and all multiplications are in \(\mathbb{Z}_{N_{e+1}}^*\)).

**Claim 5.4.** Let \(N > 1\) be an odd integer and \(e \geq 1\) be a positive integer. Then:
1. The triple representation of $E_{N,e}(0) = r^{N^e}$ (where $r \in R \mathbb{Z}_{N^e+1}^*$) is uniformly distributed over a set of the form $(P', \ast, C_1')$, where $P'$ is a subgroup of $P$ and $C_1'$ is a subgroup of $C_1$. If $N$ is a valid DJ key, then we have $P' = \{0\}$ and $C_1' = C_1 = \{0\}$, in which case $r^{N^e}$ is uniformly distributed over $(0,*,0)$. On the other hand, if $C_1$ is nontrivial then $C_1'$ is strictly contained in $C_1$.

2. For every $m \in \mathbb{Z}_{N^e}$ the ciphertext $E_{N,e}(m)$ is uniformly distributed over $(m \cdot \bar{g} + P', \ast, C_1')$, where $\bar{g} \in P$ is as in Lemma 5.3 and $P', C_1'$ are as above.

**Proof.** To prove (1), note that $r^{N^e}$ can be written (in additive triple notation) as $N^e \cdot (p,c_0,c_1)$, where $p \in R, P_0 \in R C_0, c_1 \in R C_1$ and the latter three choices are independent. Noting that $|C_0|$ is coprime to $N$ (and therefore also to $N^e$), it follows that $N^e c_0$ is uniform over $C_0$. The characterization for a valid DJ key $N$ follows from Example 5.2 and Claim 5.3. When $C_1$ is nontrivial we have $gcd(|C_1|, N) > 1$, in which case $C_1'$ a strict subset of $C_1$. Part (2) follows directly from (1), since $E_{N,e}(m) = m \cdot (\bar{g}, 0, 0) + E_{N,e}(0)$.

We now show how to exploit an invalid key $N$ to mount an explicit attack against the CDS-based construction from Section 5.5.1. We consider a degenerate case of a branching program over a single input variable, which consists of two terminal nodes and one internal node. In this case, the client should send a single ciphertext from $\mathbb{Z}_{N^e+1}^*$. We show that a query $c \in (\mathbb{Z}_{N^e+1}^* \setminus \text{support}(E_{N,e}(\mathbb{Z}_{N^e})))$ sent by a malicious client can be used to learn both output values.

**Example 5.4.** Let $N = p_1 p_2$ and $e = 1$ as in Example 5.3 (for instance, one can use $p_1 = 5$ and $p_2 = 11$). Recall that in this case $\mathbb{Z}_{N^2}^*$ is isomorphic to $\mathbb{Z}_N \times (\mathbb{Z}_2 \times \mathbb{Z}_{p_1-1}) \times \mathbb{Z}_{p_1}$, and consider the (badly formed) query $c = (0, 0, 1)$ passed to the OT. We first observe that the answer returned by $A_{OT}$ will reveal extra information. More concretely, the OT answer on a pair of bits $(s_1,s_2)$ is distributed according to $c^{s_1} \cdot (E_{N,1}(1) \cdot c^{-1})^{s_2} \cdot E_{N,1}(0)$. Since for this $N$ an encryption $E_{N,1}(0)$ is uniformly distributed over $(0,*,0)$, the answer (in triple notation) is uniformly distributed over $(s_2 \cdot \bar{g}, *, s_1 - s_2)$, and thus allows to recover both $s_1$ and $s_2$. This alone would not be a problem if the CDS gadget described in Section 5.5.1 would at least partially hide a secret $s \in R \mathbb{Z}_N$ on input $c$. Unfortunately, using the construction from [1] for well-formed DJ keys (Construction 5.2 below), would completely reveal the secret $s$, when applied under the condition that $e$ encrypts 0. Specifically, the ciphertext $\mu$ returned by the atomic CDS with randomizer $\rho$ will be uniform in $(s \cdot \bar{g}, *, \rho)$, where $s, \rho \in \mathbb{Z}_N^e$. Since $\text{ord}_P(\bar{g}) = N$, the secret $s$ can be uniquely recovered from $\mu$. Intuitively, the information stored at the third component does not interfere with the information about $s$ as long as we let the first component of $c$ be 0 (as we do here). We note that this attack can be generalized to work for other $N$’s such that $gcd(N, \phi(N)) > 1$ and other values of $e$.

In order to protect the server against maliciously formed keys, we rely on the following observation: for any odd $N$, positive integer $e$ and $c \in \mathbb{Z}_{N^e+1}^*$, the ciphertext $c' = c^{N^e T}$, where $T = \log N$, belongs to the “harmless” subgroup $(*, *, 0)$. A similar observation was used (in a different way) by Kalai [82] in the context of constructing one-round OT based on Paillier’s cryptosystem.

**Lemma 5.4.** Let $N > 1$ be an odd integer and $e \geq 1$ be a positive integer. Let $T \overset{\triangle}{=} \lceil \log_3(N) \rceil$. Then

1. For every $x \in \mathbb{Z}_{N^e+1}^*$ we have $x^{NT} \in (*, *, 0)$.
2. For \( r \in \mathbb{Z}_{N^e+1}^* \) we have \( r^{N^e+T} \in_E (0, *, 0) \).

**Proof.** It follows from Lemma 5.3 that \( |C_1| \leq \phi(S) < S \leq N \). We conclude that for any prime divisor \( p_i \) of \( |C_1| \), the highest power of \( p_i \) dividing \( N \) is no bigger than \( T \). Recalling that all prime divisors of \( |C_1| \) divide \( N \) (by definition of \( C_1 \)), we conclude that \( |C_1| \) divides \( N^T \) from which (1) follows. As to (2), since \( |P| = N^e+1/S \) and \( |C_1| \) divides \( N^T \) (by (1)), they both divide \( N^e+T \), and so \( P^{N^e+T} = C_1^{N^e+T} = \{0\} \). On the other hand, \( |C_0| \) is coprime to \( N \), and so \( C_0^{N^e+T} = C_0 \).

In our final protocol, the server will convert every ciphertext \( c \) sent by the client to a corresponding ciphertext \( c' = c^{N^T} \) which is guaranteed to be harmless. The OT and CDS implementations will be carefully adapted so that correctness is preserved, that is, the honest client will still be able to recover \( P(x) \).

The following is an extension of Observation 5.1 to general (possibly malformed) keys, which will be used to prove that the optimization of sending queries only for the highest level in the malicious setting does not compromise server privacy in the final protocol.

**Lemma 5.5.** For every odd integer \( N > 1, e > e' \geq 1, i \in \mathbb{Z} \), and \( c \) such that \( \psi(c) \in (i\tilde{g}, *, 0) \mathbb{Z}_{N^{e+1}}^* \), it holds that \( \psi(c \mod N^{e+1}) \in (i\tilde{g}^e, *, 0) \mathbb{Z}_{N^{e+1}}^* \) (\( \tilde{g}, \tilde{g} \) are as in Lemma 5.3 applied to \( \mathbb{Z}_{N^{e+1}}^* \), and adding a tag signifies we consider the group \( \mathbb{Z}_{N^{e+1}}^* \)).

**Proof.** It is sufficient to prove the claim for \( e' = e - 1 \), and the result follows for all \( e' < e \) by induction. Let \( c' = c \mod N^e \in \mathbb{Z}_{N^e}^* \) (\( c' \) is indeed an element of \( \mathbb{Z}_{N^e}^* \), since otherwise \( c \) would not be coprime to \( N^{e+1} \)). Given \( c \in (i\tilde{g}, *, 0) \mathbb{Z}_{N^{e+1}}^* \), we apply the inverse isomorphism \( \psi^{-1} \), and learn that \( c \) is of the form \( g^i r \mod N^{e+1} \), where \( \text{ord}_{\mathbb{Z}_{N^{e+1}}}(r) \) is coprime to \( N \) (by properties of \( \psi \), and structure of \( F \)). Clearly, \( g^i r \mod N^{e+1} = \psi^{-1}(\psi(c \mod N^e) = (i\tilde{g}^e, 0, 0) + \psi'(r \mod N^e) \). Applying Claim 5.2 to \( r, N, e \), \( \text{ord}_{\mathbb{Z}_{N^e}}(r \mod N^e) \) must be coprime to \( N \) (or else \( \text{ord}_{\mathbb{Z}_{N^{e+1}}}(r) \) would not be as well). We conclude that \( \psi(r \mod N^e) \) is in \((0, *, 0)\), and the Observation follows.

**High level solution description.** Our high level solution relies on the fact that for any key (except for possibly an efficiently verifiable set), the group \( \langle *, *, C'_1 \rangle \), where \( C'_1 \) is as in Claim 5.4, is a “harmless” subgroup, in the sense that if each query \( c \) belongs to this subgroup, the protocol from the previous section (with insignificant modifications) reveals no extra information. By Claim 5.4, this group contains support\( (E_{N,e}(\mathbb{Z}_{N^e})) = \langle g \rangle, C'_1 \rangle \) as a subgroup, so it poses a weaker restriction on the inputs than the construction in Section 5.5.1. In a nutshell, re-randomization by \( r^{N^e} \) takes care of attacks as in Example 5.4, by “erasing” the third component, and the CDS takes care of information leaks from the first component. A solution under this assumption (and possibly malformed keys), is described in Section 5.5.2.1.

However, there is still a gap between \( \langle *, *, C'_1 \rangle \), and \( \langle *, *, * \rangle \), as \( C'_1 \) is not necessarily \( \{0\} \), and in particular, an attack similar to that in Example 5.4 would still work. How do we ensure that the encryptions \( c_i \) sent indeed belong to the harmless subgroup? It turns out that given \( c \in \mathbb{Z}_{N^e}^* \), it’s possible to efficiently compute a related \( c' \in \langle *, 0 \rangle \subseteq \langle *, C'_1 \rangle \), knowing only \( N, e \). More specifically, given the query sequence \( c = c_1, c_2, \ldots \), we will compute an augmented sequence \( c' = c'_1, c'_2, \ldots \), where each \( c'_i \) is simply \( c_i^{N^T} \) (which is in \( \langle *, 0 \rangle \) by Lemma 5.3), and feed \( c' \), rather than \( c \) to the OT and CDS. The OT and CDS implementations will be carefully adapted so that correctness is preserved, that is, the honest client will be able to recover \( P(x) \). In particular, we will make sure that \( c_i^{N^T} \) contains information about the encrypted message \( m \) by choosing \( N^T < \text{order}(\tilde{g}, 0, 0) = N^e \).
5.5.2.1 An intermediate conditional privacy result

As explained in the overview above, in this section we explain how to achieve server’s privacy for arbitrarily malformed keys, while assuming the encryptions sent by the client fall in a certain subgroup of \( \mathbb{Z}_{N,e+1}^* \). More specifically, we describe the (slight) modification in the AIR atomic CDS construction, which allows the Construction from Section 5.5.1 to remain server private conditioned on the fact that all \( c_i \)'s sent belong to the ‘harmless’ subgroup \((\ast, \ast, C'_1)\). Although the result proven in this subsection holds for any \( e > 0 \), we assume \( e \geq T \) for simplicity, so \( C'_1 = \{0\} \). The analysis can be easily generalized to work for \( e > 0 \).

The restriction \( e \geq T \) is required for the final construction providing unconditional server privacy described in Section 5.5.2.2, so it is justified to make this simplification here, since it is only present for modular exposition on the full solution (with no assumptions), completed in the next section, where we show how to get rid of the assumption that \( c_i \in (\ast, \ast, C'_1) \) (basically, by raising the \( c_i \)'s to the power of \( N^T \)).

We start with extending the analysis of the OT construction under the assumption (no changes are required in the implementation).

Extending the analysis of Construction 5.1 to malformed keys

At this stage, we use Construction 5.1 as is. As we have seen already in the trusted setup model (see Example 5.1), the OT is not server private even according to Definition 5.8. Instead, the following ‘conditional’ notion of server privacy holds.

**Claim 5.5.** Given an odd number \( N > 1 \), \( e > T \), \( c \in E_{N,e}(b) \) for some \( b \in \{0, 1\} \), and \( s_0, s_1 \in \{0, 1\}^\tau \), \( A_{\text{OT}}((N, e), c, s_0, s_1) \) is distributed identically to \( E_{N,e}(s_1 - b) \).

**Proof.** The output of \( A_{\text{OT}} \) is distributed as \( E_{N,e}(0)c^{s_0}(E_{N,e}(1)c^{-1})^{s_1} \). Since \( c \in E_{N,e}(b) \) for some \( b \in \{0, 1\} \), it is uniformly distributed over \( E_{N,e}(b(s_0 - s_1) + s_1) = E_{N,e}(s_1 - b) \). 

This server privacy guarantee can be viewed as a strengthening of the server privacy of Strong-OT (Definition 5.7), since the output depends only on \( s_0b^* \), where \( b^* \) is an ‘effective’ selected index, even for \( N \)’s not generated by Gen, but the query must still be an ‘encryption’ of some bit (under that malformed key). We will be able to modify the CDS implementation (Construction 5.2) to allow asserting this type of ‘equality’ conditions. Unlike the OT implementation, the CDS implementation will satisfy hiding even for inputs \( c \) outside of support(\( E_{N,e} \)).

**CDS construction** For clarity, we include the original, and the augmented (AIR) atomic CDS constructions.

**Construction 5.2.** [AIR atomic CDS - original] Given input \( N, e, c, i \), where \( N > 1 \) is odd, \( e > 0 \), \( c \in \mathbb{Z}_{N,e+1}^* \), \( i \in [N^e] \),

- \( \rho \xleftarrow{\$} [N^e]; s \xleftarrow{\$} [N^e], h \leftarrow c^{-1} \cdot E_{N,e}(i) \).
- Output \( \mu = h^\rho \cdot E_{N,e}(s) \).

---

8That is, when considering an evaluation of a simple branching program, implementing an OT, even Definition 5.8 of sender’s privacy is not satisfied. This definition is potentially weaker than required from Strong-OT, as the simulator receives the query as well.

9Note that \( N^e \) is the size of \( M_{N,e} \).
Recall that for well-formed DJ keys $N, e$, this construction satisfies (1) Correctness, in the sense that if $c$ is an encryption of $i$, then $\mu$ reveals $s$. (2) Privacy, in the sense that if $c$ is not an encryption of $i$, $H_\infty(s|\mu) > 1$ (that is, “a lot” of uncertainty about $s$ remains).

**Construction 5.3.** [AIR atomic CDS - augmented] Given input $N, e, c, i$, where $N > 1$ is odd, $e \geq T$, $c \in \mathbb{Z}_{N^{e+1}}$, $i \in [N^e]$, operate as in Construction 5.2, except for sampling $\rho$ from $[N^{e+1}]$, rather than $[N^e]$. We needed to increase the domain of $\rho$ to handle inputs $c$ in $(\ast, \ast, 0)$, rather than just in support $(E_{N,e}(\mathbb{Z}_{N^e}))$, the reason will be clarified shortly. Here we extend properties (1), (2) of the construction above so that $s$ is disclosed under the condition that $c$ is in $E_{N,e}(i)$, that is, $c$ is an ‘encryption’ of $i$ under the (possibly malformed) key pair $N, e$. More formally, the equivalents of properties (1), (2) of Construction 5.2 for possibly malformed $N$, which are satisfied by the above construction are summarized by the following lemma.

**Lemma 5.6.** Let $N, e, c, i$ be as in Construction 5.3. We let $EQ(N, e, c, i)$ denote a predicate assuming 1 iff $c$ is in $E_{N,e}(i)$. (1) If $N, e$ is a valid DJ key pair, and $EQ(N, e, c, i) = 1$, then $s$ can be efficiently recovered from $\mu$. (2) Consider the joint distribution $(s, \mu)$. Assume $c \in (\ast, \ast, 0)$ such that $EQ(N, e, c, i) = 0$, then $H_\infty(s|\mu = \mu_0) \geq 1$ for any possible (for that input) output $\mu_0$.

**Proof.** Property (1) holds since if the client is honest, $\mu$ is precisely an encryption of some $s \in M_{N,e}$, and $s$ can be uniquely recovered by decrypting, as before. This is not changed by increasing the domain of $\rho$ $N$-fold, as follows directly from Claim 5.4. We turn to proving property (2). Let $c \in (\ast, \ast, 0)$, such that $EQ(N, e, c, i) = 0$. We note that for any fixed $s$, $\mu$ is uniform over $(sg + \rho h + P', \ast, 0)$, where $(\tilde{h}, r', 0)$ for some $r'$ represents $\psi(\tilde{E}_{N,e}(s)c^{-1})$, and $P'$ is as in Claim 5.4. This follows by combining Claim 5.4.1, with the assumption that $c \in (\ast, \ast, 0)$. Denote $\mu = (\tilde{\mu}, r'', 0)$. Note that due to rerandomization, only the identity of the coset of $(P', \ast, 0)$ that $\mu$ belongs to carries information about $s$. Wlog, we assume $P' = 0$ (since the additive factor of $P'$ is independent of everything else, it can only hide information about $s$). By definition, $(\tilde{g}, 0, 0), (\tilde{h}, 0, 0) \in (P, 0, 0)$, which is cyclic (isomorphic to $\mathbb{Z}_{N^{e+1}/S}$). By the assumption $EQ(N, e, c, i) = 0$, that is, $c \notin E_{N,e}(i)$, we obtain $h \notin E_{N,e}(0)$, and thus $\tilde{h} \neq 0$. Now, recall $ord(g) = ord((\tilde{g}, 0, 0)) = N^e$. Since $\tilde{h} \neq 0$, $ord(\tilde{h}, 0, 0)$ is a non-trivial divisor of $|P|$ (which equals $N^{e+1}/S$). We conclude that $p = gcd(ord((\tilde{g}, 0, 0)), ord((\tilde{h}, 0, 0))) > 1$. For any given output $\mu$, applying Claim 5.1, to the group $(P, 0, 0)$, and $(\tilde{g}, 0, 0), (\tilde{h}, 0, 0), (\tilde{\mu}, 0, 0)$, we conclude that there are $p$ distinct pairs $(s, \rho) \in [N^e] \times [ord(\tilde{h})]$ for which this $\tilde{\mu}$ is obtained, in which each value of $s$ appears in exactly one pair, and where $p$ is the smallest prime divisor of $N$. Since the algorithm samples $s$ uniformly from $[N^e]$, and $p$ uniformly from $[N^{e+1}]$, which is divisible by $ord((\tilde{h}, 0, 0))$, each solution pair is sampled the same (non-zero) number of times, so we obtain that for all possible $\mu_0$, $H_\infty(s|\mu = \mu_0) = log(p) \geq log 3 > 1$, and the result follows. \qed

CDS under monotone formulas over the atomic predicates $EQ(\cdot)$, which ‘inherits’ the same (limited) form of condition (2), can be implemented based on Construction 5.3 using the techniques from [1, 59] (we extend the construction of CDS for monotone formulas of “atomic” conditions as in Section 5.5.1). More formally, given $k$, multiple values $c_1, \ldots, c_0 \in \mathbb{Z}_{N^{e+1}}^*$, and a monotone formula $C$ over predicates $y_{i,v}$, for some $v \in [N^e]$, which assume 1 iff $EQ(N, e, c, v) = 1$, condition (1) states that if $N, e$ is a valid key pair, and $c_1, \ldots, c_0$ satisfies a set of $y_{i,v}$’s, so that $C$ evaluates to 1, then $s$ can be efficiently recovered from $\mu$. Condition (2), states that if $c_1, \ldots, c_0$ does not satisfy a set of $y_{i,v}$’s, such that $C$ evaluates to 1, then $\mu$ can be simulated from $N, e, c, |s|$ alone, with distance $\epsilon(k)$, where $\epsilon(k)$ is a negligible function (to facilitate
the use of this CDS for our application, and obtain statistical server privacy, we also “enhance” the privacy requirement so that only negligible information about s is revealed).

Revising the question of increasing the domain of \( \rho \), we see that had we sampled \( \rho \) from \( \mathbb{Z}_{N e} \), some of the solutions \((s, \tilde{\rho})\) could not be sampled (since \( \text{ord}(h') \) may be up to \( N^{e+1}/S \), which is possibly larger than \( N^{e} \)), and we could theoretically get a single reachable pair of solutions \((s, \tilde{\rho})\) while there is more than one solution (over \([N^{e}] \times [\text{ord}(h')]\)), and uniquely recover \( s \).

**Remark 5.7.** We rely on the fact that a multiple \((N^{e+1})\) of \(|P|\) can be efficiently computed from \( N, e \) alone (even for malformed \( N \)), in order to be able to uniformly sample the set of solutions. However, we could manage even if such a multiple was not known, but rather some upper bound \( B \) on it. Then, taking \([B]\) as the domain for \( \rho \), each solution would be sampled at least once, and any solution would be sampled either \( t \) or \( t + 1 \) times for some \( t \) (for \( B > |P|, H_{\infty}(s|\mu = \mu_0) \) will approach the bound on the entropy obtained by our construction).

**Putting things together.** The main protocol here is as outlined in Section 5.5.1, but using an augmented version of CDS for equality conditions (using Construction 5.3 instead of Construction 5.2), as explained above. The OT implementation remains unchanged. We do not formally prove the server privacy of this version of the main protocol, but we could readily fill in the details, and obtain a protocol for evaluating branching programs on encrypted data, whose communication complexity is comparable to that obtained for the trusted setup model, which satisfies a weaker notion of Definition 5.9, quantifying over ‘harmless’ pairs \((N, c)\) rather than arbitrary \( N, c \) (in particular, the restriction is only on \( c \)).

At large, the CDS ensures (almost) nothing is learned in cases when the OT may leak information. That is, if the conditions for hiding of the CDS do not hold, then there exists \( x^* \in \{0, 1\}^n \), such that each \( c_i \) is in \( E_{N,e}(x^*_i) \), and we can simulate the output of Eval from \( k, N, c, P(x^*) \) similarly to the way it’s done in the proof of server privacy of the (generic) protocol in Figure 5.1 in the semi-honest model (relying on Claim 5.5). Otherwise (then the OT may reveal extra information), the CDS insures that (almost) all information about the original output is lost. Note that all our lemmas have a precondition that \( N \) is odd, which can be efficiently verified. Also note that in order to make the optimization of sending only queries for the highest level, and computing the rest by modular reduction without compromising server privacy, we need to further restrict the domain of the \( c_i \)’s to be in \((*,*,0)\), so we can apply Lemma 5.5.

### 5.5.2.2 The final construction

By this point, we have achieved a ‘conditional’ form of server privacy, and it is still possible to cheat by sending \( c \) outside of \((*,*,C'_1)\) (see Example 5.4). To overcome this problem, we do not let the client generate the queries used as input to the OT and CDS single-handedly (since he may cheat), but rather jointly with the server. In other words, given the query \( c = (c_1, \ldots, c_n) \), the server applies a certain transformation to \( c \), obtaining \( c' = (c'_1, \ldots, c'_n) \), where all \( c'_i \)’s are in \((*,*,0) \subseteq (*,*,C'_1)\), and feeds them to the CDS and OT during the main protocol’s execution (the high-level structure of which remains mostly unchanged). We also modify the OT and the CDS constructions accordingly, in a way that the protocol remains correct. The transformation applied is simply \( \eta(x) = x^{N P} \). The transformation will be incorporated into the implementation of the OT and the CDS, that is, the high-level construction remains as
before, while the CDS and OT constructions first raise the input ciphertexts to the $N^T$th power, and operate on the modified ciphertexts. We first define the augmented OT procedure.

**Construction 5.4.** [Strong-OT (malicious setting)] Modify Construction 5.1 as follows.

1. $A_{OT'}(N, q, s_0, s_1)$:
   - Infer $e$ from $q$.
   - Compute an output $a$ as in the original construction.
   - Output $a' = a^{N^T}$.

2. $Q_{OT'}(N, 1^k, 1^t, 1^r, i)$: Once $e$ is fixed, operate as before. The output of $A_{OT'}$ is then an encryption of $N^T \cdot s_i$. This reduces the effective domain of the values $s_i$ that can be uniquely decrypted from $\mathbb{Z}_{N^e}$ to $\mathbb{Z}_{N^{e-T}}$. Therefore, to allow unique decryption, we choose $e$ as the minimal integer, such that $N^{e-T} > 2^T$ (rather than $N^e > 2^T$). This implies a mild increase in the expansion parameters $\alpha, \beta$ (relatively to the original construction). Namely, $\beta(k, t, \tau) = \tau + O(k \cdot T) = \tau + O(k^2)$, and $\alpha(k, t, \tau) = t(\tau + O(k^2))$ (compared to $\beta(k, t, \tau) = \tau + O(k), \alpha(k, t, \tau) = t(\tau + O(k))$ in the semi-honest and honest-setup settings).

3. $D_{OT'}((p_1, p_2), a)$: Infers $e$ from $a$, and outputs $D_{(p_1, p_2), e(a)} / N^T$, where the division is done over $\mathbb{Z}$.

It’s easy to see that the OT remains correct (since we select $e$ to be sufficiently large, the honest client uniquely recovers $s_i$). As to server privacy, the following holds.

**Claim 5.6.** Given an odd number $N > 1$, $e > 0$, $c \in \mathbb{Z}_{N^{e+1}}^*$ such that $c^{N^T} \in E_{N,e}(b)^{N^T}$ for some $b \in \{0, 1\}$, and $s_0, s_1 \in \{0, 1\}^T$, $A_{OT'}((N, e), c, s_0, s_1)$ is distributed identically to $(N^Ts_1-b\tilde{g}, *, 0)$.

**Proof.** $A_{OT'}((N, e), c, s_0, s_1) = A_{OT}((N, e), c, s_0, s_1)^{N^T} = (c^{s_0}(E_{N,e}(1) \cdot c^{-1})^{s_1} \cdot E_{N,e}(0))^{N^T} = (c^{s_0-s_1})^{N^T} \cdot E_{N,e}(s_1)^{N^T}$. Since $c^{N^T} \in E_{N,e}(b)^{N^T}$, $A_{OT'}((N, e), c, s_0, s_1)$ is uniform over $E_{N,e}(N^T(s_1 + b(s_0 - s_1)))^{N^T} \subseteq (N^T(s_1-b)\tilde{g}, *, 0)$, where the last transition follows by Lemma 5.4.2. 

It remains to show how to modify Construction 5.3.

**Construction 5.5.** For odd $N > 1$, $e > T, c \in \mathbb{Z}_{N^{e+1}}^*, i \in [N^{e-T}]$, we operate as in Construction 5.3, with the following modifications.

- Run Construction 5.3, with the difference that $s$ is sampled from $[N^{e-T}]$ (rather than $[N^e]$), to obtain $\mu$. The reason is that $\mu^{N^T}$ only contains information about $s \mod N^{e-T}$, even if $N$ is well-formed, and $EQ(N, e, c, i) = 1$, so this is required to maintain correctness. The domain of $i$ is restricted for the same reason.

- Output $\mu^{N^T}$ instead of $\mu$ ($\mu$ is computed as before).

The construction satisfies the following equivalent of Lemma 5.6.
Lemma 5.7. Let $N, e, c, i$ be as in Construction 5.5. Let $EQ'(N, e, c, i)$ denote a predicate assuming 1 iff $c^{NT} \in (iNT \tilde{g}, *, 0)$. (1) If $N, e$ is a valid DJ key pair, then $s$ can be efficiently recovered from $\mu$. (2) If $N > 1$ is odd, and $c^{NT}$ is not in $(iNT \tilde{g}, *, 0)$, then the joint distribution $(s, \mu)$, satisfies that for any possible $\mu_0$, $H_\infty(s|\mu = \mu_0) \geq 1$.

Proof. Property (1) is obvious from the construction, and the fact that $\text{ord}(g) = N^e$ for valid $N$, (by the correctness of DJ). As to property (2), let $c \in (*, *, 0)$, such that $EQ'(N, e, c, i) = 0$. We note that for any fixed $s, \rho, \mu$ is uniform over $(N^T(s\tilde{g} + \rho\tilde{h}), *, 0)$, where $(\tilde{h}, r', f')$ for some $r', f'$ represents $\psi(\tilde{E}_{N,e}(s)c^{-1})$. This follows directly from Lemma 5.4. Denote $\mu$ by $(\tilde{\mu}, r'\tilde{h}, 0)$. Since $\text{ord}(g) = N^e$, $\text{ord}(N^T \tilde{g}, 0, 0) = N^{e-T}$, which is divisible by $N$ since $e > T$. Since $c^{NT} \notin (iNT \tilde{g}, *, 0)$, $h^{NT}$ must be in $(N^T j, *, 0)$, where $j \neq i\tilde{g}$ (again, by Lemma 5.4), so $\tilde{h} \neq 0$. As in the proof of Lemma 5.6, its order $p > 1$ is a proper divisor of $N$. Thus, $(N^T \tilde{g}, 0, 0)$, and $(N^T \tilde{h}, 0, 0)$ belong to the cyclic group $(N^T P, 0, 0)$, by Claim 5.1, there are $p$ distinct pairs $(s, \rho)$ in $[N^{e-T}] \times \text{ord}(h')$, where each value of $s$ appears in exactly one pair. Since the algorithm samples $s, \rho$ from $[N^{e-T}] \times [N^{e+1}]$, and $\text{ord}((N^T \tilde{h}, 0, 0))$ divides $N^{e+1}$, for all $\mu_0$, every possible value of $s$ is sampled the same, non-zero, number of times. We conclude that $H_\infty(s|\mu = \mu_0) = \log(p) > 1$.

CDS over formulas of such equality conditions is constructed as described in Section 5.5.2.1, where $EQ(\cdot)$ is replaced by $EQ'(\cdot)$.

We are now ready to formally state the modified protocol, and prove its privacy and correctness.

Construction 5.6. [Main protocol - arbitrary malicious setting] We modify the construction described for the trusted setup setting as follows. Use Construction 5.4 as the OT implementation, and use a CDS based on Construction 5.5, to disclose the output $a$ under the condition $C = \bigwedge_{i=1}^n (y_i, 0 \lor \bar{y}_i, 1)$. Additionally, Eval checks that $N$ is an odd $2k$-bit number $> 1$. If the check does not pass, output 0. Otherwise run the high level protocol as before, in particular, we make the optimization of computing the queries for lower levels by modular reduction. The $c_i$’s are interpreted as elements of $\mathbb{Z}_{N^e+1}^*$, where the $e_j$’s corresponding to each level are as determined by Construction 5.4 on $\beta(\cdot)(k, 2, 1)$.

We prove the following theorem.

Theorem 5.3. Assuming DCRA [43], Construction 5.6 is a protocol for evaluating a binary branching program of length $\ell$ and of arbitrary size on an encrypted input of length $n$, with a total communication of $O(poly(k)n\ell^2)$ bits (where $k$ is a security parameter). The protocol provides client privacy as well as statistical size hiding server privacy according to Definition 5.9 (as noted for the construction in the trusted setup model, we can obtain $O(poly(k)n\ell)$ communication complexity if we settle for computational server privacy).

Proof. As before, client privacy follows from the semantic security of DJ. Correctness follows by correctness of the OT, and condition (1) satisfied by Construction 5.5 (Lemma 5.7). We show a simulator $\text{Sim}$ as in Definition 5.9, that proceeds as follows.

- On input $(1^k, \ell, n, N, c)$, run the validity checks as in Eval.
• If the checks do not pass, output 0, and set $x^*$ can be set to $\overline{0}$, to receive $v = U(P, x^*)$. (In this case, $x^*$ can be set to an arbitrary fixed value, as $U(P, x^*)$ is not actually used to produce the simulator’s output).

• If the checks pass, consider the sequence $e_1, \ldots, e_\ell$ specifying the groups in which to operate on each level (as computed in Eval). Check whether the $c_i$’s satisfy the CDS condition.

  - If they do, for each $i \in [n]$, let $x_i^* = i$ if $c_i^{NT} \in (iNT \cdot \overline{g}, \overline{0}) (i \in \{0, 1\})$. Let $v = U(P, x^*)$. Compute an iterative encryption of $v$. Namely, let $\lambda_0 = v$, and for $0 < j \leq \ell$, let $\lambda_j \leftarrow E_{N,e_j}(\lambda_{j-1})^{NT}$.

  - Otherwise (CDS condition is not satisfied by $c$), fix $\lambda_\ell = \overline{0}$ (or the proper size). Fix $x^* = \overline{0}$ (again, $x^*$ can be arbitrary here).

• Finally, output $\lambda_\ell \oplus s, \mu$, where $s$ is the secret disclosed via the CDS derived from Construction 5.5 under $C = \bigwedge_{i=1}^n (y_i, 0 \vee y_i, 1)$, and $\mu$ is the CDS output.

When comparing the output of Eval on input, $(1^k, \ell(n), n, N, c, P)$, and a corresponding output of the simulator on $(1^k, \ell(n), n, N, c, v = U(P, x^*))$, there are three possible cases.

1. If the validity checks on $N, c$ (in Eval) do not pass, then both the simulator and Eval in the real interaction output 0, and the simulation is perfect (for any $v$, e.g., for $v = 0$).

2. If the validity checks pass, and the conditions for (2) in the CDS do not hold for the formula $C = \bigwedge_{i=1}^n (y_i, 0 \vee y_i, 1)$ (that is, $N$ is odd, and each $c_i^{NT}$ is either in $(0, *, 0)$, or $(N^T \cdot \overline{g}, *, 0)$), then the ‘effective’ input $x^*$ extracted from $c$ by Sim is such that the output $a$ of Eval$(1^k, N, c, P)$ before applying the CDS, is distributed identically to $\lambda_\ell$ in $\text{Sim}(1^k, N, c, P(x^*))$, as we prove in the following. Finally, applying the CDS on identical distributions, result in identical distributions, and the result follows. Similarly to the proof of server privacy in the semi-honest setting, consider the path $v_0, v_1, \ldots, v_\ell$ (starting from the root) induced in $P$ by the assignment $x^*$. It holds that $\lambda_0 = P(x^*) = \text{label}(v_\ell)$. Note that each $c_{i,j}$ computed by modular reduction satisfies $c_{i,j}^{NT} \in (x^*_i)^{NT} \cdot \overline{g}, *, 0)$ for all levels $j$ (due to Lemma 5.5). Intuitively, for each $i$, the $c_{i,j}$’s are ‘consistent’ over the levels. Back to our inductive argument, traversing the path from the leaves up, $\text{label}(v_{j-1})$ is obtained from $\text{label}(v_j)$ as described in Construction 5.1, where the OT input $c_{i,j}$ is according to the label $x_j$ of that node. By Claim 5.6, $A_{\text{OT}}(N, e_{\ell-j}, c, \text{label}(v_j), \text{label}(v_0))$ (here $v_0$ denotes the other child of $v_\ell$, and $\text{label}(v_j)$ is at position $x^*_j$) is distributed identically to $E_{N,e_{\ell-j}}(\text{label}(v_j))^{NT}$, which is in turn distributed identically to $\lambda_{\ell-j+1}$. This follows by the inductive hypothesis, and the fact that the $e_j$’s are calculated by the simulator and the real execution in the same manner.

3. Assume the validity checks pass, but the conditions for (2) do hold (that is, there exists some $i \in [n]$, such that $c_i^{NT}$ is not in $(bN^T \cdot \overline{g}, *, 0)$ for some $b \in \{0, 1\}$). Then, by Lemma 5.7, $(s, \mu)$, where $\mu$ is the CDS output, and thus also $(\lambda \oplus s, \mu)$, in the real execution and the simulation is at distance at most $\epsilon(k)$ from the simulated distribution $S(1^k, N, c, e, |a|) \mu$, for $a = \lambda_\ell, \text{label}(v_0)$ respectively, which are of the same length, and $S$ is the simulator guaranteed for the CDS.\(^\text{10}\) We conclude that the

\(^{10}\)Although we do not state condition (2) in terms of simulation, it can easily be restated, and proved in this form.
distance between the output of the real and simulated execution is at most $2\epsilon(k)$, which is negligible as required.

Here we consider statistical server privacy, and the communication complexity is $O(n\ell^2\text{poly}(k))$ in this case. As mentioned in the previous section, if we settle for computational server privacy, the communication complexity may be reduced to $O(n\ell\text{poly}(k))$.

It is interesting to note that the OT construction of Kalai in [82] in the context of Paillier also implicitly relies on a variant of CDS under the condition ‘$c$ is in $\text{Enc}_N(0)$’, where $N$ is possibly malformed. Her CDS variant extends techniques of smooth projective hashing by Cramer and Shoup, for sets specified by malformed keys $N$, and has a different structure. It allows to disclose secrets over conditions of the form $c \in (0, *, 0)$, for arbitrary $N$, and $e = 1$ (any such CDS can be used to disclose secrets under conditioned as $c \in (i \cdot \tilde{g}, *, 0)$ for any $i$), for a variant of the Paillier cryptosystem. As mentioned before, this construction can be readily extended to work over (a variant of) the DJ cryptosystem. Unlike our implementation, it can work for $e < T$ (in particular, it is described for $e = 1$), and allows to disclose secrets from $\mathbb{Z}_{N^e}$, rather than $\mathbb{Z}_{N^e-T}$. Interestingly, it uses the idea of raising to to $N^T$th power in a different manner. On the other hand, client privacy of protocols based on that variant of Paillier/DJ encryption rely on the GRS assumption (additionally to the DCRA assumption).
Bibliography


חישוב בטוח_BUILDING BLOCKS אינטראקציה

ענת פסקין-אֵרנייבסקקי
חישוב בטוח בمريינמווט אינטראקטיבי

יboro על מתרק

לשם מוליית חלקי על הדיוויוט של הבולט התנור
רשקור לפלולוססיה

ענה פסקורג'יריאבסק

هجוג נסנט הטכנולוג - מכון טכנולוגי לישראל
אלול תש"ע"ב ספטמבר 2012
המחקר נעשה ברחית פיתוח מחשב יובל שי ופרופסור איל קושלביץ' בפקולטה למדעי המחשב.

אני מודע לigansי על התמיכה הכספית והדיגית ב_tweetsחולים.
תקציר

והם שחקנים במיתון של הפונקציה \( f \) \( (x_1, \ldots, x_n) \) שהופע רובים \( n \) שחקנים ובר בצורת \( \text{MPC} \) (Multiparty Computation). מקורה בחפץ \( f \) \( (x_1, \ldots, x_n) \) החופץ \( f \) \( (x_1, \ldots, x_n) \) שחקנים מעוניינים להריץ לחישוב משותף של פונקציה פומבית \( f \) \( (x_1, \ldots, x_n) \) שחקנים המעוניינים להריץ חישוב המשותף של הפונקציה \( f \) \( (x_1, \ldots, x_n) \) \( f \) \( (x_1, \ldots, x_n) \) שחקנים המעוניינים להריץ חישוב המשותף של הפונקציה \( f \) \( (x_1, \ldots, x_n) \) \( f \) \( (x_1, \ldots, x_n) \) \( f \) \( (x_1, \ldots, x_n) \) \( f \) \( (x_1, \ldots, x_n) \) \( f \) \( (x_1, \ldots, x_n) \) \( f \) \( (x_1, \ldots, x_n) \) \( f \) \( (x_1, \ldots, x_n) \) \( f \) \( (x_1, \ldots, x_n) \) \( f \) \( (x_1, \ldots, x_n) \) \( f \) \( (x_1, \ldots, x_n) \) \( f \) \( (x_1, \ldots, x_n) \) \( f \) \( (x_1, \ldots, x_n) \) \( f \) \( (x_1, \ldots, x_n) \) \( f \) \( (x_1, \ldots, x_n) \) \( f \) \( (x_1, \ldots, x_n) \) \( f \) \( (x_1, \ldots, x_n) \) \( f \) \( (x_1, \ldots, x_n) \) \( f \) \( (x_1, \ldots, x_n) \) \( f \) \( (x_1, \ldots, x_n) \) \( f \) \( (x_1, \ldots, x_n) \) \( f \) \( (x_1, \ldots, x_n) \) \( f \) \( (x_1, \ldots, x_n) \) \( f \) \( (x_1, \ldots, x_n) \) \( f \) \( (x_1, \ldots, x_n) \) \( f \) \( (x_1, \ldots, x_n) \) \( f \) \( (x_1, \ld ...)
Broadcast

The correct broadcast protocol for security with selective abort is

\[ t = \frac{2}{3} n \]

where \( n \) is the number of players. With this protocol, if a player aborts, all other players learn that something went wrong, and the protocol

\[ t = \frac{2}{3} n \]

ensures that at least one player learns the correct broadcast value.

Select abort

In a selective abort protocol, players can choose to abort the protocol at any time. The broadcast protocol ensures that at least one player learns the correct broadcast value, even if other players abort.

Poly

Poly protocols are protocols for polynomial functions. In a Poly protocol, players compute polynomials on their inputs and send the results to the server, which then combines them to compute a function of the inputs.

Randomizing polynomials

Randomizing polynomials are a technique used in secure multi-party computation to protect against passive attacks. In a randomizing polynomial protocol, players compute polynomials on their inputs and send the results to the server, which then combines them to compute a function of the inputs. The server thenrandomizes the output to protect against passive attacks.

security with selective abort

In a security protocol with selective abort, players can choose to abort the protocol at any time. The protocol ensures that at least one player learns the correct broadcast value, even if other players abort.

Technion - Computer Science Department - Ph.D. Thesis PHD-2012-16 - 2012

[52] [56] [74] [77]
In the case of shovels, unlike with computers, the program
exponentially increases the number of computer
operations. The result is that the program is
exponentially longer.

The result is that the program is
exponentially longer.