Hiding the Input-Size in Secure Two-Party Computation*

Yehuda Lindell† Kobbi Nissim‡ Claudio Orlandi§

Abstract

In the setting of secure multiparty computation, a set of parties wish to compute a joint function of their inputs, while preserving properties like *privacy*, *correctness*, and *independence of inputs*. One security property that has typically not been considered in the past relates to the *length* or *size* of the parties inputs. This is despite the fact that in many cases the size of a party's input can be confidential. The reason for this omission seems to have been the folklore belief that, as with encryption, it is impossible to carry out *non-trivial* secure computation while hiding the size of parties' inputs. However some recent results (e.g., Ishai and Paskin at TCC 2007, Ateniese, De Cristofaro and Tsudik at PKC 2011) showed that it is possible to hide the input size of one of the parties for some limited class of functions, including secure two-party set intersection. This suggests that the folklore belief may not be fully accurate.

In this work, we initiate a theoretical study of *input-size hiding* secure computation, and focus on the two-party case. We present definitions for this task, and deal with the subtleties that arise in the setting where there is no a priori polynomial bound on the parties' input sizes. Our definitional study yields a multitude of classes of input-size hiding computation, depending on whether a single party's input size remains hidden or both parties' input sizes remain hidden, and depending on who receives output and if the output size is hidden from a party in the case that it does not receive output. We prove feasibility and impossibility results for input-size hiding secure two-party computation. Some of the highlights are as follows:

- Under the assumption that fully homomorphic encryption (FHE) exists, there exist non-trivial functions (e.g., the millionaire's problem) that can be securely computed while hiding the input size of *both parties*.
- Under the assumption that FHE exists, *every* function can be securely computed while hiding the input size of one party, when both parties receive output (or when the party not receiving output does learn the size of the output). In the case of functions with fixed output length, this implies that *every* function can be securely computed while hiding one party's input size.
- There exist functions that cannot be securely computed while hiding both parties' input sizes. This is the first formal proof that, in general, some information about the size of the parties' inputs must be revealed.

Our results are in the semi-honest model. The problem of input-size hiding is already challenging in this scenario. We discuss the additional difficulties that arise in the malicious setting and leave this extension for future work.

Keywords: Secure two-party computation; input-size hiding

*This research was supported by the ERC project LAST.

†Bar-Ilan University, Israel. email: `lindell@biu.ac.il`.

‡Ben-Gurion University, Israel. `kobbi@cs.bgu.ac.il`. This work was carried out while at Bar-Ilan University.

§Aarhus University, Denmark. email: `orlandi@cs.au.dk`. This work was carried out while at Bar-Ilan University.

Contents

1 Introduction

Background. Protocols for secure two-party computation enable a pair of parties P_1 and P_2 with private inputs x and y, respectively, to compute a function f of their inputs while preserving a number of security properties. The most central of these properties are *privacy* (meaning that the parties learn the output $f(x, y)$ but nothing else), *correctness* (meaning that the output received is indeed $f(x, y)$ and not something else), and *independence of inputs* (meaning that neither party can choose its input as a function of the other party's input). The standard way of formalizing these security properties is to compare the output of a real protocol execution to an "ideal execution" in which the parties send their inputs to an incorruptible trusted party who computes the output for the parties. Informally speaking, a protocol is then secure if no real adversary attacking the real protocol can do more harm than an ideal adversary (or simulator) who interacts in the ideal model [GMW87, GL90, MR91, Bea91, Can00]. In the 1980s, it was shown that any two-party functionality can be securely computed in the presence of semi-honest and malicious adversaries [Yao86]. Thus, this stringent definition of security can actually be achieved.

Privacy and size hiding. Clearly, the security obtained in the ideal model is the most that one can hope for. However, when looking closer at the formalization of this notion, it is apparent that the statement of privacy that "nothing but the output is learned" is somewhat of an overstatement. This is due to the fact that the size of the parties' inputs (and thus also the size of the output) is assumed to be known (see Section 3.1 for a discussion on how this is actually formalized in the current definitions). However, this information itself may be confidential. Consider the case of set intersection and companies who wish to see if they have common clients. Needless to say, the number of clients that a company has is itself highly confidential. Thus, the question that arises is whether or not it is possible to achieve secure computation while hiding the size of the parties' inputs. We stress that the fact that input sizes are revealed is not a mere artifact of the definition, and all standard protocols for secure computation indeed assume that the input sizes are publicly known to the parties.

The fact that the input size is always assumed to be revealed is due to the folklore belief that, as with encryption, the length of the parties' inputs cannot be hidden in a secure computation protocol. In particular, the definition in [Gol04, Sec. 7.2.1.1] uses the convention that both inputs are of the same size, and states *"Observe that making no restriction on the relationship among the lengths of the two inputs disallows the existence of secure protocols for computing any* non-degenerate *functionality. The reason is that the program of each party (in a protocol for computing the desired functionality) must either depend only on the length of the party's input or obtain information on the counterpart's input length. In case information of the latter type is not implied by the output value, a secure protocol* cannot afford *to give it away"*. In the same way in [HL10, Sec. 2.3] it is stated that *"We remark that some restriction on the input lengths is unavoidable because, as in the case of encryption, to some extent such information is always leaked."*. It is not difficult to see that there exist functions for which hiding the size of both inputs is impossible (although this has not been formally proven prior to this paper). However, this does not necessarily mean that "non-degenerate" or "interesting" functions cannot be securely computed without revealing the size of one or both parties' inputs.

State of the art. The first work to explicitly refer to hiding input size is that of zero-knowledge sets [MRK03], in which a prover commits to a set S and later proves statements of the form $x \in S$ or $x \notin S$ to a verifier, without revealing anything about the cardinality of S. Zero-knowledge sets are an interesting instance of size-hiding reactive functionality, while in this work we only focus on non-reactive computation (i.e., secure function evaluation).

Ishai and Paskin [IP07] also explicitly refer to the problem of hiding input size, and construct a homomorphic encryption scheme that allows a party to evaluate a branching program on an encrypted input, so that the *length* of the branching program (i.e., the longest path from the initial node to any terminal node) is revealed but nothing else about its size. This enables partial input-size hiding two-party computation by having one party encode its input into the branching program. In particular this implies a secure two-party private set intersection protocol where the size of of the set of one of the two parties is hidden.

Ateniese et al. [ACT11] constructed the first (explicit) protocol for private set-intersection that hides the size of one of the two input sets. The focus of their work is on efficiency and their protocol achieves high efficiency, in the random oracle model. The construction in their paper is secure for semi-honest adversaries, and for a weaker notion of one-sided simulatability when the adversary may be malicious (this notion guarantees privacy, but not correctness, for example). In addition, their construction relies on a random oracle.

Those works demonstrate that interesting, non-degenerate functions *can* be computed while at least hiding the input size of one of the parties, and this raises a number of fascinating and fundamental questions:

> *Can input-size hiding be formalized in general, and is it possible to securely compute many (or even all) functions while hiding the input size of one of the parties?*
>
> *Are there any interesting functions that can be securely computed while hiding both parties' inputs sizes?*

Before proceeding, we remark that in many cases it is possible to hide the input sizes by using *padding*. However, this requires an a priori upper bound on the sizes of the inputs. In addition, it means that the complexity of the protocol is related to the maximum possible lengths and is thus *inherently inefficient*. Thus, this question is of interest from both a theoretical point of view (is it possible to hide input size when no a priori upper bound on the inputs is known and so its complexity depends only on each party's own input and output), and from a practical point of view. In this paper we focus on theoretical feasibility, and therefore we do not consider side-channel attacks that might be used to learn additional information about a party's input size e.g., by measuring the response time of that party in the protocol, but we hope that our results will stimulate future work on more efficient and practical protocols.

Our results. In this paper, we initiate the theoretical study of the problem of input-size hiding two-party computation. Our main contributions are as follows:

- *Definition and classification:* Even though some input-size hiding protocols have been presented in the literature, no formal definition of input-size hiding generic secure computation has ever been presented. We provide such a definition and deal with technical subtleties that relate to the fact that no a priori bound on the parties' input sizes is given (e.g., this raises an issue as to how to even define polynomial-time for a party running such a protocol). In addition, we observe that feasibility and infeasibility depend very much on which party receives output, whether or not the output-size is revealed to a party not receiving output, and whether one party's input size is hidden or both. We therefore define a set of classes of input-size hiding variants, and a unified definition of security. We also revisit the standard definition where both parties' input sizes are revealed and observe that the treatment of this case is much more subtle than has been previously observed. (For example, the standard protocols for secure computation are *not* secure under a definition of secure computation for which both parties receive output if their input sizes are equal, and otherwise both parties receive $\perp$. We show how this can be easily fixed.)

- *One-party input-size hiding:* We prove that in the case that one party's input size is hidden and the other party's input size is revealed, then *every* function can be securely computed in the presence of semi-honest adversaries, when both parties receive either the output or learn the output size (or when the output size can be upper bounded as a function of one party's input size). This includes the problem of set intersection and thus we show that the result of [ACT11] can be achieved without random oracles and under the full ideal/real simulation definition of security. Our protocols use fully homomorphic encryption [Gen09] (we remark that although this is a very powerful tool, there are subtleties that arise in attempting to use it in our setting). This is the first general feasibility result for input-size hiding.

  We also prove that there exist functionalities (e.g., unbounded input-length oblivious transfer) that *cannot* be securely computed in the presence of semi-honest adversaries while hiding one party's input size, if one of the parties is not supposed to learn the output size. This is also the first formal impossibility result for input-size hiding, and it also demonstrates that the size of the output is of crucial consideration in our setting. (In the standard definition where input sizes are revealed, a fixed polynomial upper-bound on the output size is always known and can be used.)

- *Two-party input-size hiding:* We prove that there exist functions of interest that can be securely computed in the presence of semi-honest adversaries while hiding the input size of *both* parties. In particular, we show that the greater-than function (a.k.a., the millionaires' problem) can be securely computed while hiding the input size of both parties. In addition, we show that the equality, mean, median, variance and minimum functions can all be computed while hiding the size of both parties' inputs (our positive result holds for any function that can be efficiently computed with polylogarithmic communication complexity). To the best of our knowledge, these are the first examples of non trivial secure computation that hides the size of both parties' inputs, and thus demonstrate that non-degenerate and interesting functions can be securely computed in contradiction to the accepted folklore. We also prove a general impossibility result that it is impossible to hide both parties' input sizes for any function (with fixed output size) with randomized communication complexity $\Omega(n^{\varepsilon})$ for some $\varepsilon > 0$. Combined with our positive result, this is an almost complete characterization of feasibility.

- *Separations between size-hiding variants:* We prove separations between different variants of size-hiding secure computation, as described above. This study shows that the issue of size-hiding in secure computation is very delicate, and the question of who receives output and so on has a significant effect on feasibility.

Our results provide a broad picture of feasibility and infeasibility, and demonstrate a rich structure between the different variants of input-size hiding. We believe that our results send a clear message that input-size hiding is possible, and we hope that this will encourage future research to further understand feasibility and infeasibility, and to achieve input-size hiding with practical efficiency, especially in applications where the size of the input is confidential.

Malicious adversaries – future work. In this initial foundational study of the question of size-hiding in secure computation, we mainly focus on the model of semi-honest adversaries. As we will show, many subtleties and difficulties arise already in this setting. In the case of malicious adversaries, it is even more problematic. One specific difficulty that arises in this setting is due to the fact that the simulator must run in time that is polynomial in the adversary. This is a problem since any input-size hiding protocol must have communication complexity that is independent of the parties' inputs sizes. Thus, the simulator must extract the corrupted party's input (in order to send it to the trusted party) even if it is very long, and in particular even if its length is not a priori polynomially bounded in the communication complexity. In order to ensure that the simulator is polynomial in the adversary, it is therefore necessary that the simulator somehow knows how long the adversary would run for. This is a type of "proof of work" for which rigorous solutions do not exist. We remark that we do provide definitions for the case of malicious adversaries. However, the problem of *constructing* input-size hiding protocols for the case of malicious adversaries is left for future work.

In a concurrent work Chase and Visconti [CV12] gave the first protocol that implements zero-knowledge sets with simulation based security against malicious adversaries under standard assumption. Their main tool is an enhanced version of universal arguments with a stronger proof-of-knowledge property. Is is not clear whether this tool can be adapted and used for general functionalities.

2 Technical Overview

In this section we provide a brief overview of the results and the techniques used through the paper together with pointers to the contents of this document.

Definitions. In Section 3 we formalize the notion of input-size hiding in secure two-party computation, following the ideal/real paradigm. As opposed to the standard ideal model, we define the sizes of the input and output values as explicit additional input/outputs of the ideal functionality and, by considering all the combinations of possible output patterns we give a complete classification of ideal functionalities. The different classes can be found in Figure 2 on the last page of this document. We consider three main classes (class 0,1 and 2) depending on how many input sizes are kept hidden (that is, in class 2 the size of both parties input is kept hidden, in class 1 the size on party's input is kept hidden, and in class 0 neither parties inputs are hidden). Even for class 0, where both input sizes are allowed to leak, we argue that our definition of the ideal world is more natural and general than the standard one (see Section 3.1). This is due to the fact that in standard definitions, it is assumed that the parties have agreed on the input sizes in some "out of band" method. As we show, this actually leads to surprising problems regarding the definition of security and known protocols. Each of the classes is then divided into subclasses, depending on what kind of information about the output each party receives (each party can learn the output value, the output size or no information about the output). As we will see, the information about the output that is leaked, and to which party, has significant ramifications on feasibility and infeasibility.

The next step on the way to providing a formal definition is to redefine the notion of a protocol that runs in polynomial time (see Definition 3.1). In order to see why this is necessary, observe that there may not exist any single polynomial that bounds the length of the output received by a party, as a function of its input. This is because the length of the output may depend on the length of the other party's input, which can vary. We provide definitions for the semi-honest and the malicious cases in Definition 3.2 and 3.3.

Class 1 – positive and negative results. In Section 4.1 we show how every function can be computed while hiding the input size of one party, if both parties are allowed to learn the *size of the output* (or its actual value). The idea behind our protocol is very simple, and uses fully homomorphic encryption (FHE) with circuit privacy (see Appendix A): One party encrypts her input x under her public key and sends it to the other party, who then uses the homomorphic properties in order to compute an encryption of the output $f(x, y)$ and sends the encrypted result back. Due to circuit privacy, this does not reveal any information about the length of $|y|$ and therefore size-hiding is achieved. Despite its conceptual simplicity, we observe that one subtle issue arises. Specifically, the second party needs to know the length of the output (or an upper bound on this length) since it needs to construct a circuit computing f on the encrypted x and on y. Of course, given $|x|$ and $|y|$ it is possible to compute such an upper bound, and the ciphertext containing the output can be of this size. Since P_2 knows $|x|$ and y it can clearly compute this bound, but when P_1 receives the encrypted output it would learn the bound which could reveal information about $|y|$. We solve this problem by having the parties first compute the exact size of the output, using FHE. Then, given this exact size, they proceed as described above.

It turns out that this simple protocol is in fact optimal for class 1 (even though P_2 learns the length of the output $f(x, y)$), since it is in general impossible to hide the size of the input of one party and the size of the output at the same time. In Section 5.3 we prove that two natural functions (oblivious transfer with unbounded message length and oblivious pseudorandom-function evaluation) cannot be securely computed in two of the subclasses of class 1 where only one party receives output, and the party not receiving output is not allowed to learn the output size. The intuition is that the size of the transcript of a size-hiding protocol must be independent of the size of one of the inputs (or it will reveal information about it). But, as the length of the output grows with the size of the input, we reach a contradiction with incompressibility of (pseudo)random data.

Class 2 – positive and negative results. In this class, both of the parties' input sizes must remain hidden; as such, this is a much more difficult setting and the protocol described above for class 1 cannot be used. Nevertheless, we present positive results for this class and show that every function that can be computed insecurely using a protocol with *low communication complexity* can be compiled into a size-hiding secure two party protocol. The exact requirements for the underlying (insecure) protocol are given in Definition 4.5 and the compilation uses FHE and techniques similar to the one discussed for class 1 above. Interesting examples of functions that can be securely computed while hiding the size of both parties input using our technique include statistical computations on data such as computing the mean, variance and median. With some tweaks, known protocols with low communication complexity for equality or the greater-than function can also be turned into protocols satisfying our requirements (see Section 4.3.4).

As opposed to class 1, we do not have any general positive result for class 2. Indeed, in Theorem 5.1 we show that there exist functions that *cannot* be securely computed while hiding the input size of both parties. Intuitively, in a size-hiding protocol the communication complexity must be independent of the input sizes and therefore we reach a contradiction with lower-bounds in communication complexity. Examples of interesting functions that cannot be computed in class 2 include the inner product, hamming distance and set intersection functions.

Separations between classes. In Section 5.2 we show that even in class 2, the output size plays an important role. Specifically, we show that there exist functions that can be computed in class

2 only if both parties are allowed to learn the output size. Finally, in Section 5.4 and 5.5, more separations between classes and subclasses are showed. We highlight that, perhaps surprisingly, class 2 is not a subset of class 1. That is, there exist functions that cannot be computed in some subclasses of class 1 that can be securely computed in class 2. These results demonstrate that the input-size hiding landscape is rich, as summarized in Table 1 in Section 6.

3 Definitions – Size-Hiding Secure Two-Party Computation

In this section, we formalize the notion of input-size hiding in secure two-party computation. Our formalization follows the ideal/real paradigm for defining security due to [Can00, Gol04]. Thus, we specify the security goals (what is learned by the parties and what is not) by describing appropriate ideal models where the parties send their inputs to an incorruptible trusted party who sends each party exactly what information it is supposed to learn. The information sent to a party can include the *function output* (if it is supposed to receive output), the other party's *input-length* (if it is supposed to learn this), and/or the *length of the function output* (this can make a difference in the case that a party does not learn the actual output). We will define multiple ideal models, covering the different possibilities regarding which party receives which information. As we will see, what is learned and by whom makes a big difference to feasibility. In addition, in different applications it may be important to hide different information (in some client/server "secure set intersection" applications it may be important to hide the size of both input sets, only the size of one the input sets, or it may not be important to hide either). Our definitions are all for the case of *static adversaries*, and so we consider only the setting where one party is honest and the other is corrupted; the identity of the corrupted party is fixed before the protocol execution begins.

The function and the ideal model: We distinguish between the *function f* that the parties wish to compute, and the *ideal model* that describes how the parties and the adversary interact and what information is revealed and how. The ideal model type expresses the security properties that we require from our cryptographic protocol, including which party should learn which output, what information is leaked to the adversary, which party is allowed to learn the output first and so on. In our presentation, we focus on the two-party case only; the extension to the multiparty setting is straightforward.

3.1 Revisiting the Standard Definition – No Size Hiding

In this section we review the standard way that input sizes are dealt with and observe that there are important subtleties here which are typically ignored.

We begin by reviewing how previous definitions dealt with the fact that the lengths of both parties' inputs is revealed by known secure protocols. Beyond motivating some of our definitional choices, we will argue that previous definitions can and should be strengthened, even when the input sizes are revealed.

The definition of security [Gol04, Def. 7.2.2] deals with the fact that input lengths are revealed by only guaranteeing security when the inputs are of the same length. Clearly this is a simplifying convention. However, it has the artifact that no security at all is required when $|x| \neq |y|$. Thus, a protocol that instructs a party to send its input in the clear to the other if it turns out they have different input lengths is actually secure by this definition (which is clearly not the intention of the definition).

An alternative approach that is also proposed by [Gol04] (see the paragraph on "partial functionalities" in Section 7.2.1.1) is to securely compute a function of the form

$$f'(x,y) = \begin{array}{ll} \big(f(x,y), f(x,y)\big) & \text{if } |x| = |y| \\ (\perp, \perp) & \text{if } |x| \neq |y| \end{array}.$$

This solves the aforementioned problem since the parties learn nothing if their inputs are of different lengths. The use of this functionality requires that the parties have *a priori* information about each other's input length (unless it suffices for them to receive $\perp$ when this is not the case). This is usually justified saying that the parties need anyway to agree on a set of common parameters (e.g., the security parameter) before starting the computation. However, when considering malicious parties (that can replace their inputs in the ideal model), defining the functionality in this way introduces an *input length independence* problem. Specifically, a malicious party can choose its input based on the length of the other party's input. An example of where this can be problematic is in Yao's classic millionaires' problem. Consider a party who wishes (for whatever reason) to have the output being that its salary is greater if the honest party's salary is greater than \$1,000,000, and otherwise it wishes the output to be that its salary is smaller. This can be carried out by first seeing if the honest party's input x is of length 20 bits or less, and then choosing its input y to be $0^{|x|}$ if $|x| < 20$, and choosing its input y to be $1^{|x|}$ if $|x| \geq 20$. Arguably, such behavior is undesirable. Of course, it is possible to solve this problem by *padding* the input lengths to some maximum bound. However, this is only a partial solution, and may be unsatisfactory (depending on the application) for two reasons. First, there may not exist a reasonable upper bound on the input length of the parties, and the process of coming up with such an upper bound may also reveal information that enables the adversary to achieve partial input length dependence. Second, padding to a maximum introduces an *inherent inefficency* in the computation.

We therefore advocate an alternative way of defining security when both parties inputs sizes are revealed, as follows:

$$f''(x,y) = \Big((1^{|y|}, f(x,y)), (1^{|x|}, f(x,y))\Big).$$

Observe that security is obtained for all input lengths, and the input length of the other parties' input is revealed only *after* each party has provided its own input. (In our actual formulation, the trusted party will receive x and y from each party, and will send the appropriate input lengths together with the function output.)

Observe that the security guaranteed by computing $f''(x,y)$ is *incomparable* to the security guaranteed by computing $f'(x,y)$. This is due to the fact that, on the one hand, $f''(x,y)$ does not suffer from the input-length dependence problem of $f'(x,y)$, as described. However, on the other hand, $f'(x,y)$ does not reveal the parties' input lengths if they are not the same. In addition, $f'(x,y)$ ensures that a party uses an input of a prescribed length, and not an input that is shorter or longer. This can be important in some settings (e.g., in a private set-intersection protocol over a small domain, a cheating party could input a database containing every possible element and thus completely learn the honest party's input set).[1] Thus, depending on the application, one may prefer the f' or the f'' formulations.

We stress that the standard protocols for secure computation in the presence of semi-honest adversaries (e.g., Yao [Yao86] and GMW [GMW87]) are *not* secure under the f' formulation since

[1]It is actually possible to ensure that a party uses a prescribed input length also using the f'' formulation. This can be achieved by redefining the function f so that each party inputs the allowed input-length of the other party, and setting the output to be $\perp$ in case mismatching input lengths are used.

they reveal the input size in the case that these sizes are different. In order to solve this problem, the parties need to first run a secure comparison protocol on the input sizes (encoding the sizes in binary and padding with zeroes to a string that is superlogarithmic in length in order to hide the size). Then, if the inputs are the same length they proceed, and if not then they halt without learning anything except for this fact. Regarding the f'' formulation, although it is true that the standard protocols are secure for semi-honest adversaries, this is *not* the case for security in the presence of malicious adversaries (e.g., in GMW [GMW87]). This is because a malicious adversary can choose its input after seeing the first message from the honest party, which reveals its input length. Nevertheless, it is possible to modify the GMW protocol (which starts by each party committing to its input using a perfectly-binding commitment) by having the parties first commit to their inputs using a statistically-hiding, length-hiding commitment, and proving knowledge of the committed value. The parties then commit to their inputs using a perfectly-binding (thus length revealing) commitment, and provide zero-knowledge proofs that the committed values are the same. They then proceed to run the original GMW protocol. This achieves the required notion since the input lengths are not revealed before the parties are committed to their input values. (The unmodified protocols of Yao and GMW, when compiled via GMW to be secure in the presence of malicious adversaries do securely compute a weaker formulation in which both parties receive $f(x,y)$ in case the inputs are of the same length, and receive the inputs lengths in case they are not.)

3.2 Classes of Size Hiding

We define three classes of size hiding, differentiated by whether neither party's input size is hidden, one party's input size is hidden or both parties input sizes are hidden (note that the class number describes how many input sizes are kept hidden: 0, 1 or 2):

1. *Class 0:* In this class, the input size of both parties is revealed (See Section 3.1);
2. *Class 1:* In this class, the input size of one party is hidden and the other is revealed. There are a number of variants in this class, depending on whether one or both parties receive output, and in the case that one party receives output depending on whose input size is hidden and whether or not the output size is hidden from the party not receiving output.
3. *Class 2:* In this class, the input size of both parties' inputs are hidden. As in Class 1 there are a number of variants depending on who receives output and if the output size is kept hidden to a party not receiving output.

We now turn to describe the different variants/subclasses to each class. Due to the large number of different subclasses, we only consider the more limited case that when both parties receive output, then they both receive the *same* output $f(x,y)$. When general feasibility results can be achieved, meaning that any function can be securely computed, then this is without loss of generality [Gol04, Prop. 7.2.11]. However, as we will see, not all classes of input-size hiding yield general feasibility; the study of what happens in such classes when the parties may receive different outputs is left for future work.

Subclass definitions:

0. *Class 0:* We formalize both the f' and f'' formulations from Section 3.1. In both formulations, we consider only the case that both parties receive the function output $f(x,y)$. There is no

need to consider the case that only one party receives $f(x,y)$ separately here, since general feasibility results hold and so there is a general reduction from the case that both receive output and only one receives output. In addition, we add a strictly weaker formulation where both parties receive $f(x,y)$ if $|x| = |y|$, and otherwise receive only the input lengths. We include this since the standard protocols for secure computation are actually secure under this formulation. The subclasses are:

(a) *Class 0.a:* if $|x| = |y|$ then both parties receive $f(x,y)$, and if $|x| \neq |y|$ then both parties receive $\perp$

(b) *Class 0.b:* if $|x| = |y|$ then both parties receive $f(x,y)$, and if $|x| \neq |y|$ then P_1 receives $1^{|y|}$ and P_2 receives $1^{|x|}$

(c) *Class 0.c:* P_1 receives $(1^{|y|}, f(x,y))$ and P_2 receives $(1^{|x|}, f(x,y))$

In Section 3.1 it is shown that every functionality can be securely computed in classes 0.a, 0.b and 0.c.

1. *Class 1:* We consider five different subclasses here. In all subclasses, the input-size $1^{|x|}$ of P_1 is revealed to P_2, but the input-size of P_2 is hidden from P_1. The different subclasses are:

   (a) *Class 1.a:* both parties receive $f(x,y)$, and P_2 learns $1^{|x|}$ as well

   (b) *Class 1.b:* only P_1 receives $f(x,y)$, and P_2 only learns $1^{|x|}$

   (c) *Class 1.c:* only P_1 receives $f(x,y)$, and P_2 learns $1^{|x|}$ and the output length $1^{|f(x,y)|}$

   (d) *Class 1.d:* P_1 learns nothing at all, and P_2 receives $1^{|x|}$ and $f(x,y)$

   (e) *Class 1.e:* P_1 learns $1^{|f(x,y)|}$ only, and P_2 receives $1^{|x|}$ and $f(x,y)$

2. *Class 2:* We consider three different subclasses here. In all subclasses, no input-sizes are revealed. The different subclasses are:

   (a) *Class 2.a:* both parties receive $f(x,y)$, and nothing else

   (b) *Class 2.b:* only P_1 receives $f(x,y)$, and P_2 learns nothing

   (c) *Class 2.c:* only P_1 receives $f(x,y)$, and P_2 learns the length of the output $1^{|f(x,y)|}$

See Figure 2 (at the last page of this document) for a graphic description of the above (we recommend referring back to the figure throughout). We stress that the question of whether or not the output length $1^{|f(x,y)|}$ is revealed to a party not receiving $f(x,y)$ is of importance since, unlike in standard secure computation, a party not receiving $f(x,y)$ or the other party's input size cannot compute a bound on $1^{|f(x,y)|}$. Thus, this can make a difference to feasibility. Indeed, as we will see, when $1^{|f(x,y)|}$ is not revealed, it is sometimes impossible to achieve input size-hiding.

When considering *symmetric functions* (where $f(x,y) = f(y,x)$ for all x,y), the above set of subclasses covers *all* possible variants for classes 1 and 2 regarding which parties receive output or output length. This is due to the fact that when the function is symmetric, it is possible to reverse the roles of the parties (e.g., if P_2's input-length is to be revealed to P_1, then by symmetry the parties can just exchange roles in class 1). We focus on symmetric functions in this paper (as described in Footnote 3 the non-symmetric case is not so different with respect to feasibility; we leave the additional complexity of non-symmetric functions for future work.)

We remark that P_1's input-length and the output-length are given in unary, when revealed; this is needed to give the simulator enough time to work in the case that one party's input is much shorter than the other party's input and/or the output length.

3.3 The Ideal Models

We are now ready to define the ideal model. In fact, we actually define a different ideal model for each class, representing what information is revealed in each class. We denote by $\text{IDEAL}^{A.b}$ the ideal model corresponding to Class $A.b$ (e.g., $\text{IDEAL}^{2.a}$ denotes the ideal model in which both parties receive $f(x, y)$ and nothing else). We only consider security with abort here, and so the difference between classes that give outputs to both parties or just one is not due to the question of fairness, but about the amount of information that the computation reveals. We stress that in the setting of input-size hiding, even if we only consider semi-honest corruptions, the case that one party receives $f(x, y)$ and the other learns nothing, is not equivalent to the case that both parties receive $f(x, y)$.

Denote the participating parties by P_1 and P_2 and let $i \in \{1, 2\}$ denote the index of the corrupted party, controlled by an ideal-model adversary $\mathcal{S}$. An ideal execution for a (single-output) function $f : \{0, 1\}^* \times \{0, 1\}^* \to \{0, 1\}^*$ proceeds as follows:

Inputs: Let x denote the input of party P_1, and let y denote the input of party P_2. The adversary $\mathcal{S}$ also has an auxiliary input denoted by z.

Send inputs to trusted party: The honest party sends its received input to the trusted party. The corrupted party controlled by $\mathcal{S}$ may either send its received input, or send another input *of any length* to the trusted party. Denote the pair of inputs sent to the trusted party by (x', y') (note that since an honest party does not change its input, it holds that if $i = 2$ then $x' = x$, and if $i = 1$ then $y' = y$).

Trusted party sends output to adversary: The trusted party computes $f(x', y')$ and sends the corrupted party its prescribed output; this output can include $f(x', y')$ or $1^{|f(x',y')|}$, and the input-length of the honest party, depending on the class.

Adversary instructs trusted party to continue or halt: $\mathcal{S}$ sends either continue or abort to the trusted party. If it sends continue, the trusted party sends the honest party its prescribed output, depending on the class. Otherwise, if $\mathcal{S}$ sends abort, the trusted party sends abort to the honest party.

Outputs: The honest party outputs whatever it received from the trusted party. The corrupted party outputs nothing, and $\mathcal{S}$ outputs any function of its view.

Let $f : \{0, 1\}^* \times \{0, 1\}^* \to \{0, 1\}^*$ be a (possibly randomized) polynomial-time computable functionality, let $\mathcal{S}$ be an ideal adversary, and let $i \in \{1, 2\}$ be the index of the corrupted party. Then, the ideal execution of f in class $A.b$ on inputs (x, y), auxiliary input z to $\mathcal{S}$ and security parameter κ, denoted by $\text{IDEAL}^{A.b}_{f,\mathcal{S}(z),i}(x, y, \kappa)$, is defined as the output pair of the honest party and the adversary $\mathcal{S}$ from the above ideal execution.

3.4 The Real Model

In the real model with a protocol π (specifying instructions for the parties P_1 and P_2) and real-model *malicious* adversary $\mathcal{A}$, the honest party runs the specified protocol and outputs whatever it is instructed to output by the protocol. The adversary controls the corrupted party and can send any messages that it wishes, irrespective of the protocol specification, and then outputs any arbitrary function of its view. Without loss of generality, one can assume that the adversary outputs its view only. We denote by $\text{REAL}_{\pi,\mathcal{A}(z),i}(x, y, \kappa)$ the output pair of the adversary $\mathcal{A}$ controlling P_i

and the honest party from a real execution of π, where $\mathcal{A}$ has auxiliary input z, the parties have inputs x and y, and the security parameter is κ.

The above is the standard description of the real model for secure computation, and this is unchanged in the setting of input size-hiding computation. However, since the parties may have inputs of arbitrary length (that are not a priori bounded by any given polynomial), we have to redefine the notion of a protocol that runs in polynomial time. In order to see why this is a problem, observe that there may not exist any single polynomial that bounds the length of the output received by a party, as a function of its input. This is because the length of the output may depend on the length of the other party's input, which can vary. This issue becomes more severe in the case of malicious adversaries, since the adversary may choose the length of the input that it uses and is not limited to the length of the input written on the corrupted party's input tape. Due to this, we say that a protocol is polynomial time if:

1. When both parties follow the protocol specification, each party's running-time is polynomial in the lengths of its input and output, and the security parameter (note that a party's running time does not depend on the other party's input length, if this is hidden).

2. When one party is maliciously corrupted, then the running time of the honest party is polynomial in the lengths of its input and the security parameter, and the running-time of the corrupted party. Observe that in this case, there is no single polynomial that bounds the running time of an honest party in every execution. However, for each polynomial-time adversary, there is a polynomial that bounds the running-time of the honest party in any execution with this adversary.

We stress that a party's input and output actually depends on the *class* that we consider. This is due to the fact that if the protocol reveals the length of P_1's input to P_2, then P_2 must be able to run at least this long. As such, for $i \in \{1,2\}$, we denote by $\textsc{output}_i^{A.b}(x,y)$ the prescribed output of party P_i in class $A.b$ as specified in Section 3.2; e.g., in class 1.a we have that $\textsc{output}_2^{1.a}(x,y) = (1^{|x|}, f(x,y))$. Furthermore, denote by $\textsc{time}_{P_i}^{\pi}(x,y,\kappa)$ the running time of P_i in π when both parties follow the protocol specification, with respective inputs (x,y) and security parameter κ (we consider separate input and security parameter tapes, and assume that all honest parties and adversaries have the same value κ on their security parameter tape). Finally, denote by $\textsc{time}_{\mathcal{A}}(z,\kappa)$ the running time of $\mathcal{A}$ on auxiliary input z and security parameter κ, and by $\textsc{time}_{P_i}^{\mathcal{A}}(x,y,z,\kappa)$ the running-time of the honest party P_i running π and interacting with adversary $\mathcal{A}$.

We are now ready to formally define what it means for a protocol to be polynomial-time for a class:

Definition 3.1 (Polynomial-Time Protocol) *Let π be a two-party protocol. We say that π is* polynomial-time for class A.b *if:*

1. Honest executions: *There exists a polynomial $p(\cdot)$ such that for every $\kappa \in \mathbb{N}$ and every pair of inputs $x, y \in \{0,1\}^*$:*
$$\textsc{time}_{P_1}^{\pi}(x,y,\kappa) \le p\left(|x| + |\textsc{output}_1^{A.b}(x,y)| + \kappa\right), \quad \text{and}$$
$$\textsc{time}_{P_2}^{\pi}(x,y,\kappa) \le p\left(|y| + |\textsc{output}_2^{A.b}(x,y)| + \kappa\right),$$

2. Dishonest executions: *There exists a polynomial $q(\cdot)$ such that for every polynomial-time adversary $\mathcal{A}$, every $\kappa \in \mathbb{N}$ and all inputs $x, y, z \in \{0,1\}^*$:*

$$\text{TIME}_{P_1}^{\mathcal{A}}(x,y,\kappa) \le q\left(|x| + \kappa + \text{TIME}_{\mathcal{A}}(z,\kappa)\right), \quad \text{and}$$
$$\text{TIME}_{P_2}^{\mathcal{A}}(x,y,\kappa) \le q\left(|y| + \kappa + \text{TIME}_{\mathcal{A}}(z,\kappa)\right).$$

We remark that the latter requirement regarding dishonest executions is needed in order to prevent a polynomial-time adversary from making an honest party run in super-polynomial time (through some malicious activity).

3.5 Defining Security

Semi-honest adversaries. Security is defined in the semi-honest setting by requiring the existence of simulators $\mathcal{S}_1$ and $\mathcal{S}_2$ that can generate the view of the parties P_1 and P_2, respectively, given their input and output. The actual requirement is that the *joint* view of the simulator's output and the function output is computationally indistinguishable from the corrupted party's view and the honest party's output in a protocol execution. We define computational indistinguishability in the usual way, and denote it by $\overset{c}{\equiv}$.

Let $\text{OUTPUT}^{\pi}(x,y,\kappa)$ denote the joint output of both parties from an execution of π, and let $\text{view}_i^{\pi}(x,y,\kappa)$ denote the view of party P_i in the execution. When we write the view and OUTPUT in the same ensemble, then this refers to the view of the party and the outputs in the same execution. Likewise, when we write $\text{OUTPUT}_1^{A.b}(x,y)$ and $\text{OUTPUT}_2^{A.b}(x,y)$ in the same ensemble, this refers to the same computation of $f(x,y)$. This is of importance when f is probabilistic and so $f(x,y)$ is not determined by x, y alone. See [Gol04] for discussion.

Definition 3.2 (Security for Class $A.b$ – Semi-Honest) *Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be a functionality, and let π be a polynomial time protocol for class $A.b$. We say that π* securely computes f in class $A.b$ in the presence of semi-honest adversaries *if there exist probabilistic polynomial time-algorithms $\mathcal{S}_1, \mathcal{S}_2$ such that for every pair of polynomials $q_1(\cdot)$ and $q_2(\cdot)$,*

$$\left\{\left(\mathcal{S}_1\left(x, \text{OUTPUT}_1^{A.b}(x,y)\right), \text{OUTPUT}^{A.b}(x,y)\right)\right\}_{\kappa,x,y} \overset{c}{\equiv} \{(\text{view}_1^{\pi}(x,y,\kappa), \text{OUTPUT}^{\pi}(x,y,\kappa))\}_{\kappa,x,y}$$
$$\left\{\left(\mathcal{S}_2\left(y, \text{OUTPUT}_2^{A.b}(x,y)\right), \text{OUTPUT}^{A.b}(x,y)\right)\right\}_{\kappa,x,y} \overset{c}{\equiv} \{(\text{view}_2^{\pi}(x,y,\kappa), \text{OUTPUT}^{\pi}(x,y,\kappa))\}_{\kappa,x,y}$$

where $\kappa \in \mathbb{N}$, $x \in \{0,1\}^{q_1(\kappa)}$ and $y \in \{0,1\}^{q_2(\kappa)}$.

We stress that the distinguisher and simulators run in time that is polynomial in their input length (as revealed by the class) and 1^κ. By convention, we do not write 1^κ as part of a machine's input, as it is on a separate security parameter tape.

Malicious adversaries. Security in the case of malicious adversaries is defined via the ideal and real models described above. Loosely speaking, the definition asserts that a secure party protocol (in the real model) emulates the ideal model (in which a trusted party exists). This is formulated by saying that adversaries in the ideal model are able to simulate executions of the real-model protocol. In the definition, we refer to polynomial-time machines (adversaries and distinguisher). As above, the distinguisher and adversaries run in time that is polynomial in their input plus the security parameter κ (i.e., $\mathcal{A}$ is polynomial-time if there exists a polynomial $p(\cdot)$ such that $\mathcal{A}$ always halts within time $p(|z| + \kappa)$, where z is its auxiliary input).

Definition 3.3 (Security for Class $A.b$ – Malicious) *Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be a functionality, and let π be a polynomial time protocol for class $A.b$. We say that π* securely computes f in class $A.b$ in the presence of malicious adversaries *if for every non-uniform probabilistic polynomial-time real-model malicious adversary $\mathcal{A}$ there exists a non-uniform probabilistic polynomial-time ideal-model malicious adversary $\mathcal{S}$ such that for all polynomials $q_1(\cdot)$, $q_2(\cdot)$ and $q_3(\cdot)$,*

$$\left\{\text{IDEAL}^{A.b}_{f,\mathcal{S}(z),i}(x,y,\kappa)\right\}_{k,x,y,z} \overset{\text{c}}{\equiv} \left\{\text{REAL}_{\pi,\mathcal{A}(z),i}(x,y,\kappa)\right\}_{k,x,y,z}$$

where $\kappa \in \mathbb{N}$, $x \in \{0,1\}^{q_1(\kappa)}$, $y \in \{0,1\}^{q_2(\kappa)}$, and $z \in \{0,1\}^{q_3(\kappa)}$.

It is important to note that the simulator in the case of malicious adversaries may actually not have enough time to read the input and output of the corrupted party. This is because its running time is polynomial in $|z| + \kappa$, and this may be much shorter than the input/output in the class. Regarding reading the corrupted party's input, we can assume that z is at least as long as the corrupted party's input and so the adversary (and simulator) always has time to read this input. However, we cannot make a similar assumption regarding the output. This is because the length of the output $f(x,y)$ depends on the inputs of the honest and corrupted party. Thus, there is no a priori length that we can refer to. Due to this, we cannot give the adversary the length of the output of the class and ensure that it runs polynomial in this. We therefore have the adversary run in time polynomial in $|z| + \kappa$ and must prove security when it has and does not have time to read the output (i.e., for all polynomial-time adversaries).

4 Feasibility Results

4.1 General Constructions for Class 1.a/c/e Input-Size Hiding Protocols

In this section, we prove a general feasibility result that any function f can be securely computed in classes 1.a, 1.c and 1.e (recall that in class 1, the size of P_2's input is hidden from P_1, but the size of P_1's input is revealed to P_2). In Section 5, we will see that such a result cannot be achieved for classes 1.b and 1.d, and so we limit ourselves to classes 1.a/c/e. We begin by proving the result for class 1.c, where P_1 obtains the output $f(x,y)$, and P_2 obtains P_1's input length $1^{|x|}$ and the output length $1^{|f(x,y)|}$, and then show how a general protocol for class 1.c can be used to construct general protocols for classes 1.a and 1.e.

The idea behind our protocol is very simple, and uses fully homomorphic encryption (FHE) with circuit privacy; see Appendix A for the definition. Party P_1 begins by choosing a key-pair for an FHE scheme, encrypts its input under the public key, and sends the public key and encrypted input to P_2. This ciphertext reveals the input length of P_1, but this is allowed in class 1.c. Next, P_2 computes the function on the encrypted input and its own input, and obtain an encryption of $f(x,y)$. Finally, P_2 sends the result to P_1, who decrypts and obtains the output. Observe that this also reveals the output length to P_2, but again this is allowed in class 1.c.

Despite its conceptual simplicity, we observe that one subtle issue arises. Specifically, party P_2 needs to know the length of the output $f(x,y)$, or an upper bound on this length, since it needs to construct a circuit computing f on the encrypted x and on y. Of course, given $|x|$ and $|y|$ it is possible to compute such an upper bound, and the ciphertext containing the output can be of this size (the actual output length may be shorter, and this can be handled by having the output of the circuit include the actual output length). Since P_2 knows $|x|$ and y it can clearly compute this bound. However, somewhat surprisingly, having P_2 compute the upper bound may actually reveal information about P_2's input size to P_1. In order to see this, consider the set union functionality. Clearly, the output length is upper bounded by the sum of the length of P_1's input and P_2's input, but if P_2 were to use this upper bound then P_1 would be able to learn the length of P_2's input which is not allowed. We solve this problem by having the parties first compute the exact size of the output, using FHE. Then, given this exact size, they proceed as described above. The protocol is presented in Figure 4.1, and uses an FHE scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$. We denote by n the length $|x|$ of P_1's input, and by m the length $|y|$ of P_2's input. In addition, we denote $x = x_1, \ldots, x_n$ and $y = y_1, \ldots, y_m$.

PROTOCOL 4.1 (Class 1.c Size-Hiding for Any Functionality – Semi-Honest)

- **Inputs:** P_1 has x, and P_2 has y. Both parties have security parameter 1^κ.
- **The protocol:**
  1. P_1 chooses $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$, computes $c_1 = \mathsf{Enc}_{pk}(x_1), \ldots, c_n = \mathsf{Enc}_{pk}(x_n)$ and sends $(pk, c_1, \ldots, c_n)$ to P_2.
  2. P_2 receives $c_1, \ldots, c_n$, and constructs a circuit $\mathcal{C}_{size,y}(\cdot)$ that computes the output length of $f(\cdot, y)$ in binary (i.e., $\mathcal{C}_{size,y}(x) = |f(x,y)|$), padded with zeroes up to length $\log^2 \kappa$. Then, P_2 computes $c_{size} = \mathsf{Eval}_{pk}(\mathcal{C}_{size,y}, \langle c_1, \ldots, c_n \rangle)$, and sends c_{size} to P_1.
  3. P_1 receives c_{size} and decrypts it using sk; let ℓ be the result. Party P_1 sends ℓ to P_2.
  4. P_2 receives ℓ from P_1 and constructs another circuit $\mathcal{C}_{f,y}(\cdot)$ that computes $f(x,y)$ (i.e., $\mathcal{C}_{f,y}(x) = f(x,y)$), and has ℓ output wires. Then, P_2 computes $c_f = \mathsf{Eval}_{pk}(\mathcal{C}_{f,y}, \langle c_1, \ldots, c_n \rangle)$, and sends c_f to P_1.
  5. P_1 receives c_f and decrypts it using sk to obtain a string z.
- **Outputs:** P_1 outputs the string z obtained in the previous step; P_2 outputs nothing.

Theorem 4.2 *Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time computable function. If* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *constitutes a fully homomorphic encryption with circuit privacy, then Protocol 4.1 securely computes f in class 1.c, in the presence of a static semi-honest adversary.*

Proof: Recall that in order to prove security in the presence of semi-honest adversaries, it suffices to present simulators $\mathcal{S}_1$ and $\mathcal{S}_2$ that receive the input/output of parties P_1 and P_2, respectively, and generate their view in the protocol. The requirement is that the joint distribution of the view generated by the simulator and the honest party's output be indistinguishable from the view of the corrupted party and the honest party's output.

We begin with the case that P_1 is corrupted. Simulator $\mathcal{S}_1$ receives $(x, f(x,y))$ and prepares a uniformly distributed random tape for P_1. Then, $\mathcal{S}_1$ uses that random tape to sample $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$. Then, $\mathcal{S}_1$ computes $c_{size} = \mathsf{Enc}_{pk}(|f(x,y)|)$ padded with zeroes up to length $\log^2 \kappa$, and $c_f = \mathsf{Enc}_{pk}(f(x,y))$. Finally, $\mathcal{S}_1$ outputs the input x, the random tape chosen above, and the incoming messages c_{size} and c_f. The only difference between the view generated by $\mathcal{S}_1$ and that of P_1 in a real execution is that c_{size} and c_f are generated by directly encrypting $|f(x,y)|$ and $f(x,y)$, rather than by running Eval. However, the *circuit privacy* requirement guarantees that the distributions over these ciphertexts are statistically close.

Next, consider a corrupted P_2. Simulator $\mathcal{S}_2$ receives $(y, (1^{|x|}, 1^{|f(x,y)|}))$, and generates $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$ and $c_1 = \mathsf{Enc}_{pk}(0), \ldots, c_{|x|} = \mathsf{Enc}_{pk}(0)$. Then, $\mathcal{S}_2$ outputs y, a uniform random tape, and incoming messages $(pk, c_1, \ldots, c_{|x|}, |f(x, y)|)$ as P_2's view. The indistinguishability of the simulated view from a real view follows immediately from the regular encryption security of the fully homomorphic encryption scheme. ■

4.2 More Feasibility Results for Class 1

It is not difficult to see that given protocols for class 1.c, it is possible to obtain protocols for classes 1.a and 1.e (for class 1.a just have P_1 send the output to P_2, and the compute in class 1.e by computing a function in class 1.a that masks the output from P_1 so that only P_2 can actually obtain it). In addition, we show that with the function has a bounded output length (meaning that it is some fixed polynomial in the length of P_1's input), then any function can be securely computed in classes 1.b and 1.e as well. An important application of this is the *private set intersection problem* (observe that the size of the output is upper bounded by the size of P_1's input). We therefore obtain an analogue to the result of [ACT11] without relying on random oracles. In this section we present these extensions.

4.2.1 Securely computing classes 1.a and 1.e.

Recall that in class 1.c, party P_1 learns $f(x, y)$ and party P_2 learns $1^{|f(x,y)|}$ and $1^{|x|}$. In class 1.a the only difference is that both P_1 and P_2 obtain $f(x, y)$. Thus, a protocol for class 1.a can be obtained from Protocol 4.1 by having P_1 send $z = f(x, y)$ to P_2 after decrypting it in the last step.

In contrast, in class 1.e, the difference is that P_1 obtains only $1^{|f(x,y)|}$ and P_2 obtains the actual output $f(x, y)$ (as well as $1^{|x|}$). In order to achieve this, define the function $f'(x, (y, r)) = f(x, y) \oplus G(r)$ where $r \in \{0, 1\}^\kappa$ and G is a pseudorandom generator that stretches κ bits to $|f(x, y)|$ bits (which is also polynomial; see Definition 3.2). Then, the parties use the class 1.a protocol described above to compute f', where P_2 chooses the r part of its input uniformly at random. Finally, P_2 XORs its output with $G(r)$ in order to obtain $f(x, y)$ and outputs it. Clearly, P_1 learns only $1^{|f(x,y)|}$ since the output is masked (a simulator can just replace the value seen by P_1 by a random string and this will be indistinguishable by the pseudorandom property of the generator). Thus, this protocol securely computes f in class 1.e.

Corollary 4.3 *Let $f : \{0, 1\}^* \times \{0, 1\}^* \to \{0, 1\}^*$ be a polynomial-time computable function. If fully homomorphic encryption with circuit privacy exists, then there exist protocols for securely computing f in classes 1.a, 1.c, and 1.e, in the presence of static semi-honest adversaries.*

4.2.2 Constructions for Classes 1.b/1.d with Output-Length Bounded Functions

In many natural examples, the security offered by class 1.a/c/e might not be sufficient. Consider a scenario where a client and a server wish to compute the intersection of their sets, and the client is supposed to learn the output without the server learning *anything* (neither the size of the client's set nor the size of the output). In such a case, the client must play P_2 (in order to hide the size of its input) and the server plays P_1. However, in classes 1.a, 1.c and 1.e, party P_1 always learns either the output itself or the output size. We would therefore like to be able to compute this function in class 1.d (and in other settings class 1.b may be of interest).

In this section, we show that when the output size of f can be bounded as a function of P_1's input size only, then Protocol 4.1 can be used to securely compute f in *any* subclass of class 1. This includes the important class of functions with *fixed* output size (e.g., functions that output a single bit). Formally,

Corollary 4.4 *Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time computable function, such that there exists a polynomial $p_1(\cdot)$ so that for every x it holds that $|f(x,y)| \leq p_1(|x|)$. If fully homomorphic encryption with circuit privacy exists, then there exist protocols for securely computing f in all of class 1 (including 1.b and 1.d) in the presence of static semi-honest adversaries.*

Proof: The feasibility of computing in classes 1.a, 1.c and 1.e has already been shown in Corollary 4.3. In order to prove the result for class 1.b (where P_1 receives $f(x,y)$ only, and P_2 receives $1^{|x|}$ only), define the function $f'(x,y) = f(x,y)\|0^{p_1(|x|)-|f(x,y)|}$, where $\|$ denotes string concatenation. The parties then compute f' using Protocol 4.1 for class 1.c. Observe that P_1 learns $f(x,y)$ only from this protocol, and P_2 learns $1^{|x|}$ and $1^{|f'(x,y)|}$. However, $|f'(x,y)| = p_1(|x|)$ which can be computed from $1^{|x|}$. Thus, P_2's view can be simulated given only $1^{|x|}$, as required in class 1.b.

In order to prove the result for class 1.d (where P_1 learns nothing at all, and P_2 learns $f(x,y)$ and $1^{|x|}$), the parties securely compute f' using a protocol for class 1.e. Thus, P_1's output from the protocol is $1^{|f'(x,y)|}$, and P_2's output is $(1^{|x|}, f(x,y))$. Observing again that $|f'(x,y)| = p_1(|x|)$ we have that P_1's view can be computed given its input x only. Likewise, since P_2 receives $1^{|x|}$ it can compute $p_1(|x|)$ by itself. Thus, we have that this protocol securely computes f in class 1.d. ■

Application – private set intersection. In [ACT11], a protocol that securely computes the private set intersection function in class 1.d was presented (i.e., in their protocol the server learns nothing while the client learns the server's input size and the output). Their construction is efficient, but it relies on the random oracle model, and is only secure for "one-sided simulation". (That is, the case of a corrupted client can be simulated, but in the case of a corrupted server only privacy is guaranteed. This means that a malicious server could break the properties of correctness and independence of inputs.)

Our focus in this paper is not efficiency, and we do not construct efficient protocols. Rather, we wish to prove that input-size hiding is feasible, and in particular that it is possible to securely compute the private set intersection function while hiding the input size of the client, in the standard model. We obtain this corollary by observing that with set intersection it is possible to upper bound the size of the output knowing the size of one input. This is because $|X \cap Y| \leq |X|$. Likewise, set difference can be computed because $|X \setminus Y| \leq |X|$ (although it is not a symmetric function). Thus, set intersection and difference can be securely computed in classes 1.b and 1.c.

4.3 Feasibility for Some Functions in Class 2

In this section we prove that some non-trivial functions can be securely computed in class 2. This is of interest since class 2 protocols reveal nothing about either party's input size, beyond what is revealed by the output size. In addition, in class 2.b, nothing at all is revealed to party P_2. We start by presenting protocols for class 2.c and then discuss how these can be extended to class 2.a, and in what cases they can be extended to class 2.b.

There are functionalities that are impossible to securely compute in any subclass of class 2; see Section 5. Thus, the aim here is just to show that some functions can be securely computed; as we will see, there is actually quite a large class of such functions. We leave the question of characterizing exactly what functions can and cannot be computed for future work.

4.3.1 Class 2.c

We begin by considering class 2.c, where party P_1 receives the output $f(x,y)$ and P_2 receives $1^{|f(x,y)|}$, but nothing else is revealed. Intuitively this is possible for functions that can be computed efficiently by two parties (by an insecure protocol), with communication that can be upper bounded by some fixed polynomial in the security parameter. In such cases, it is possible to construct size-hiding secure protocols by having the parties run the insecure protocol inside fully homomorphic encryption. We formalize what we require from the insecure protocol, as follows.

Definition 4.5 (size-independent protocols) *Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$, and let π be a probabilistic protocol. We say that π is* size independent *if it satisfies the following properties:*

- Correctness: *For every pair of polynomials $q_1(\cdot), q_2(\cdot)$ there exists a negligible function μ such that for every $\kappa \in \mathbb{N}$, and all $x \in \{0,1\}^{q_1(\kappa)}, y \in \{0,1\}^{q_2(\kappa)}$: $\Pr[\pi(x,y) \neq f(x,y)] \leq \mu(\kappa)$.*
- Computation efficiency: *There exist polynomial-time interactive probabilistic Turing Machines π_1, π_2 such that for every pair of polynomials $q_1(\cdot), q_2(\cdot)$, all sufficiently large $\kappa \in \mathbb{N}$, and every $x \in \{0,1\}^{q_1(\kappa)}, y \in \{0,1\}^{q_2(\kappa)}$, it holds that $(\pi_1(1^\kappa, x), \pi_2(1^\kappa, y))$ implements $\pi(x,y)$.*
- Communication efficiency: *There exists a polynomial $p(\cdot)$ such that for every pair of polynomials $q_1(\cdot), q_2(\cdot)$, all sufficiently large $\kappa \in \mathbb{N}$, and every $x \in \{0,1\}^{q_1(\kappa)}, y \in \{0,1\}^{q_2(\kappa)}$, the number of rounds and length of every message sent in $\pi(x,y)$ is upper bounded by $p(\kappa)$.*

Observe that by computation and communication efficiency, given x, κ and a random tape r, it is possible to efficiently compute a series of circuits $\mathcal{C}^1_{P_1,\kappa,x,r}, \ldots, \mathcal{C}^{p(\kappa)-1}_{P_1,\kappa,x,r}$ that compute the next message function of $\pi_1(1^\kappa, x; r)$ (i.e., the input to the circuit $\mathcal{C}^i_{P_1,\kappa,x,r}$ is a vector of $i-1$ incoming messages of length $p(\kappa)$ each, and the output is the response of P_1 with input x, security parameter κ, random coins r, and the incoming messages given in the input). Likewise, given y, κ and s, it is possible to efficiently compute analogous $\mathcal{C}^1_{P_2,\kappa,y,s}, \ldots, \mathcal{C}^{p(\kappa)}_{P_2,\kappa,y,s}$. We stress that since the length of each message in π is bounded by $p(\kappa)$, the circuits can be defined with input length as described above. For simplicity, we assume that in each round of the protocol the parties exchange messages that are dependent only on messages received in the previous rounds (this is without loss of generality).

In addition, it is possible to generate a circuit $\mathcal{C}^{\mathsf{output}}_{P_1,\kappa,x,r}$ for computing the output of P_1 given its input and all incoming messages. As in Protocol 4.1, in order to generate $\mathcal{C}^{\mathsf{output}}_{P_1,\kappa,x,r}$ we need to know the *exact* output size (recall that using an upper bound may reveal information). Therefore, we also use a circuit $\mathcal{C}^{\mathsf{size}}_{P_1,\kappa,x,r}$ that computes the exact output length given all incoming messages; this circuit has output length $\log^2 \kappa$ (and so any polynomial output length can be encoded in binary in this number of bits) and can also be efficiently generated.

We start with class 2.c and show that if a function has a size-independent protocol, then we can securely compute the function in class 2.c. In more detail, a size-independent protocol has communication complexity that can be bound by a fixed polynomial $p(\kappa)$, for inputs of any length (actually, of length at most $\kappa^{\log \kappa}$ and so for any a priori unbounded polynomial-length inputs)[2]. Then, we can run this protocol *inside* fully homomorphic encryption; by padding all messages to

[2]Note that upper bounding the input sizes to $\kappa^{\log \kappa}$ is not a real restriction: if the adversary has enough time to read an input of this size, then it has time to break the underlying computational assumption and no secure protocol exists.

their upper bound (and likewise the number of messages), we have that nothing is revealed by the size of the ciphertexts sent. We note, however, that unlike in the protocols for class 1, in this case neither party is allowed to know the secret key of the fully homomorphic encryption scheme (since both parties must exchange ciphertexts, as in the communication complexity protocol). This is achieved by using threshold key generation and decryption, which can be obtained using standard secure computation techniques (observe that no size hiding issues arise regarding this).

PROTOCOL 4.6 (Size-Hiding Protocol for Class 2.c)

- **Inputs:** P_1 has x, and P_2 has y. Both parties have security parameter 1^κ.
- **Auxiliary input:** A size independent protocol π for f and the polynomial $p(\cdot)$ bounding the communication efficiency as in Definition 4.5.
- **The protocol:**
  1. P_1 and P_2 invoke a secure (class 0) protocol computing the functionality ThrGen for threshold FHE with inputs $1^\kappa, 1^\kappa$ (see Appendix A) and obtain a public key pk and shares of the secret key sk_1,sk_2, respectively.
  2. *P_1 and P_2 run an encrypted execution of π*: P_1 and P_2 choose random coins r and s, respectively, of the appropriate length for π. Then, for $i = 1$ to $p(\kappa)$:
     (a) P_1 generates the circuit $\mathcal{C}^i_{P_1,\kappa,x,r}$, computes $c^i_1 = \mathsf{Eval}_{pk}(\mathcal{C}^i_{P_1,\kappa,x,r}, \langle c^1_2, \ldots, c^{i-1}_2 \rangle)$, and sends c^i_1 to P_2.
     (b) P_2 generates the circuit $\mathcal{C}^i_{P_2,\kappa,y,s}$, computes $c^i_2 = \mathsf{Eval}_{pk}(\mathcal{C}^i_{P_2,\kappa,y,s}, \langle c^1_1, \ldots, c^{i-1}_1 \rangle)$, and sends c^i_2 to P_1.
  3. *P_1 and P_2 compute the exact output length:*
     (a) P_1 generates the circuit $\mathcal{C}^{\mathsf{size}}_{P_1,\kappa,x,r}$ for computing the exact output size. Then, it computes $c_{size} = \mathsf{Eval}_{pk}(\mathcal{C}^{\mathsf{output}}_{P_1,\kappa,x,r}, \langle c^1_2, \ldots, c^{p(\kappa)}_2 \rangle)$, and sends c_{size} to P_2.
     (b) P_1 and P_2 invoke a secure (class 0) protocol computing the functionality ThrDec, on respective inputs (c_{size}, sk_1) and (c_{size}, sk_2). P_1 receives the decrypted value, and P_2 receives nothing. Let t be the integer encoded in binary that is output.
  4. *P_1 and P_2 compute the output:*
     (a) P_1 generates the circuit $\mathcal{C}^{\mathsf{output}}_{P_1,\kappa,x,r}$ with t output wires. Then, P_1 computes $c_f = \mathsf{Eval}_{pk}(\mathcal{C}^{\mathsf{output}}_{P_1,\kappa,x,r}, \langle c^1_2, \ldots, c^{p(\kappa)}_2 \rangle)$, and sends c_f to P_2.
     (b) P_1 and P_2 invoke a secure (class 0) protocol computing the functionality ThrDec, on respective inputs (c_f, sk_1) and (c_f, sk_2). P_1 receives the decrypted value z, and P_2 receives nothing.
- **Outputs:** P_1 outputs z and P_2 outputs nothing.

Theorem 4.7 *Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be a function. If π is a size-independent protocol for computing f, and* ($\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval}$) *is a fully homomorphic encryption scheme, then Protocol 4.6 securely computes f in class 2.c in the presence of static semi-honest adversaries.*

Proof: We begin with the case that P_1 is corrupted. Simulator $\mathcal{S}_1$ receives $(x, f(x,y))$ and prepares a uniformly distributed random tape for P_1. Then, $\mathcal{S}_1$ chooses $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$ and simulates an execution of ThrGen with input 1^κ and output pk, sk_1, where sk_1 is just a random string of the appropriate length. Next, $\mathcal{S}_1$ generates $p(\kappa)$ encryptions of zeroes $c^1_2 = \mathsf{Enc}_{pk}(0^{p(\kappa)}), \ldots, c^{p(\kappa)}_2 =$

$\mathsf{Enc}_{pk}\left(0^{p(\kappa)}\right)$. Then, $\mathcal{S}_1$ computes $c_{size} = \mathsf{Eval}_{pk}\left(\mathcal{C}^{\mathsf{size}}_{P_1,\kappa,x,r}, \langle c_2^1, \ldots, c_2^{p(\kappa)}\rangle\right)$ and simulates an execution of ThrDec with input c_{size} and output $t = |f(x,y)|$ (encoded in binary, and padded to length $\log^2 \kappa$). Finally, $\mathcal{S}_1$ computes $c_f = \mathsf{Eval}_{pk}\left(\mathcal{C}^{\mathsf{output}}_{P_1,\kappa,x,r}, \langle c_2^1, \ldots, c_2^{p(\kappa)}\rangle\right)$ and simulates an execution of ThrDec with input c_f and output $z = f(x,y)$. Simulator $\mathcal{S}_1$ sets the view of P_1 to be the concatenation of all of the above. In order to prove that the view generated by $\mathcal{S}_1$ is indistinguishable from the view generated in a real execution, we first consider a hybrid simulator $\mathcal{S}'_1$ who is given y and works exactly like $\mathcal{S}_1$ except that the $p(\kappa)$ ciphertexts $c_1^1, \ldots, c_1^{p(\kappa)}$ are computed to be encryptions of the messages sent by P_2 in π, upon input y. Indistinguishability is derived from the CPA security of the fully homomorphic encryption scheme (observe that $\mathcal{S}_1$ and $\mathcal{S}'$ never use sk so they can work when given pk as input; thus this reduction can go through). Now, observe that the difference between the view generated by $\mathcal{S}'$ and the view of P_1 in a real protocol execution is just that $\mathcal{S}'$ simulates the executions of ThrGen and ThrDec, whereas real executions are used in the protocol. Indistinguishability thus follows directly from the security of ThrGen and ThrDec.

Next, consider a corrupted P_2. Simulator $\mathcal{S}_2$ receives $(y, 1^{|f(x,y)|})$ and prepares a uniformly distributed random tape for P_1. Then, $\mathcal{S}_2$ chooses $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$ and simulates an execution of ThrGen with input 1^κ and output pk, sk_2, where sk_2 is just a random string of the appropriate length. Next, $\mathcal{S}_2$ generates $p(\kappa)$ encryptions $c_1^1 = \mathsf{Enc}_{pk}\left(0^{p(\kappa)}\right), \ldots, c_1^{p(\kappa)} = \mathsf{Enc}_{pk}\left(0^{p(\kappa)}\right)$. Then, $\mathcal{S}_2$ computes $c_{size} = \mathsf{Enc}_{pk}\left(0^{\log^2 \kappa}\right)$ and simulates an execution of ThrDec with input c_{size} and no output for P_2. Then, $\mathcal{S}_2$ simulates P_1 sending P_2 the value $t = |f(x,y)|$ (encoded in binary, and padded to length $\log^2 \kappa$). Finally, $\mathcal{S}_2$ computes $c_f = \mathsf{Enc}_{pk}(0^t)$ and simulates an execution of ThrDec with input c_f and no output for P_2. Simulator $\mathcal{S}_2$ sets the view of P_2 to be the concatenation of all of the above. Indistinguishability here follows from similar arguments to the case that P_1 is corrupted. ■

4.3.2 Class 2.a and 2.b

In order to obtain a class 2.a protocol from a class 2.c protocol, all that is needed is for party P_1 to send the output to party P_2. In contrast, we cannot obtain class 2.b protocols using the above protocol for class 2.a since the output length is not allowed to be revealed in class 2.b. Nevertheless, if the output length is fixed (as in binary functions, for example), then the output length is already known and so this issue does not arise. We therefore have the following corollary.

Corollary 4.8 *Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be a function. If there exists a size-independent protocol for computing f, and fully homomorphic encryption schemes exist, then f can be securely computed in classes 2.a and 2.c in the presence of static semi-honest adversaries. Furthermore, if in addition to the above the output-size of f is fixed for all inputs, then f can be securely computed in class 2.b in the presence of static semi-honest adversaries.*

4.3.3 Applications – Secure Computation in Classes 2.a/b/c

The millionaires' problem, formally defined by $\mathsf{GT}(x,y) = 1$ if $x > y$ and 0 otherwise (where $x, y \in \mathbb{N}$ encoding in binary), can be computed insecurely with communication complexity that is logarithmic in the lengths of (the binary encoding of) the inputs x and y.[3] A protocol for this can be derived

[3]Observe that the greater-than function is not symmetric (recall that a function f is symmetric if $f(x,y) = f(y,x)$ for all x, y), and in this paper we focus on symmetric functions. Nevertheless, it can be made symmetric by defining

from the communication complexity protocol for GT by Nisan and Safra [Nis93, Theorem 1a]. This protocol carries out a binary search in order to find the first κ-bit block where x and y differ, and then just exchange that block. The only difference required is that the binary search carried out by [Nis93] uses an equality check on arbitrary-size blocks with error, whereas we can check equality using collision-resistant hashing. Due to its importance, and non-triviality, we describe this protocol in Section 4.3.4.

We remark that low communication protocols exist for a large variety of tasks, notably including statistical computations on data. For example, consider the case that two parties hold private databases of salaries, and wish to compute the mean, variance and median of their salaries. This is easily carried out: (a) the mean can be locally computed by each and then combined given the overall number of employees; (b) the variance can be locally computed and combined once given the mean; (c) the median can be computed with logarithmic communication complexity [KN97]. Finally, note that *equality* can also be computed with a size-independent protocol by simply having the parties exchange a hash of their inputs, using a collision-resistant hash function with output length κ.

All of the above functions have fixed output length. Therefore by Corollary 4.8:

Corollary 4.9 *Assuming the existence of fully homomorphic encryption, the greater-than, equality, mean, variance and median functions can be securely computed in classes 2.a, 2.b and 2.c, in the presence of static semi-honest adversaries.*

In order to compute the minimum function $\mathsf{min}(x, y)$ for $x, y \in \mathbb{N}$, we first compute the length of $\mathsf{min}(x, y)$; i.e., we first compute $\mathsf{min}(|x|, |y|)$. This can be carried out by first computing $\mathsf{GT}(x, y)$ and then having the party with the smaller (or equal) value send its input length. By defining the output length to be a binary encoding padded with zeroes up to length $\log^2 \kappa$, we have that the output of the "min-length" function is fixed size. Thus, it can be computed in class 2.a according to Corollary 4.8. Now, given the output length t, party P_1 can encrypt the first t bits of its input using fully homomorphic encryption, and then P_2 can use Eval to compute the actual minimum that P_1 can decrypt. This securely computes $\mathsf{min}(x, y)$ in class 2.c, and class 2.a follows trivially by having P_1 send the output to P_2. (This is not secure for class 2.b since P_2 learns the output size.) Therefore:

Corollary 4.10 *Assuming the existence of fully homomorphic encryption, the* min *function can be securely computed in classes 2.a and 2.c, in the presence of static semi-honest adversaries.*

4.3.4 An Input-Size Hiding Protocol For The Millionaires' Problem

In this section, we show how the millionaires' problem can be securely computed in class 2. Our protocol works by applying Protocol 4.6 to an insecure protocol for computing the greater-than function, with communication complexity that depends only on the security parameter and is *independent* of the actual input length. The fact that this suffices has been proven in Corollary 4.8. It therefore remains to show how it is possible to compute the greater-than function in this way. We demonstrate this by describing a variant of the communication complexity protocol of [Nis93]. We

$f((x, b_1), (y, b_2))$ to equal $\mathsf{GT}(x, y)$ if $b_1 = 0$ and $b_2 = 1$, and to equal $\mathsf{GT}(y, x)$ if $b_1 = 1$ and $b_2 = 0$, and to equal $(\perp, b_1)$ if $b_1 = b_2$. Since b_1 and b_2 are always revealed, it is possible for the parties to simply exchange these bits, and then to run the protocol for GT in the "appropriate direction", revealing the output as determined by the class.

stress that our setting is different since we require negligible error (and not constant error as in the communication complexity setting), and we can also rely on computational assumptions.

Our protocol uses a collision-resistant hash function $h : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^\kappa$ with the property that $h(0^\kappa, 0^\kappa) = 0^\kappa$. This can be constructed from any collision-resistant hash function h' by letting $h(x,y) = h'(x,y) \oplus h'(0^\kappa, 0^\kappa)$. It is easy to see that if h' is collision-resistant, then so is h.

Background. Define $\mathsf{Tree} : \{0,1\}^{\kappa 2^\ell} \to \kappa$ to be the function that takes as input a string x of length $\kappa 2^\ell$, builds a hash tree of depth ℓ from it and outputs the value at the root. More in detail, we assign a name and value to each node in the tree. Let λ be the name of the root of the tree. Then, for any node with name u, let the name of its children be $u0$ and $u1$ (thus the children of the root are named $0, 1$, their children are named $00, 01, 10, 11$, and so on). The hash tree is of depth ℓ and therefore there are 2^ℓ leaves in the tree. The values of the nodes are then computed in a bottom up fashion. Specifically, the input string x of length $\kappa \cdot 2^\ell$ is broken up into 2^ℓ blocks of length κ each; denote $x = (x_0, \ldots, x_{2^\ell - 1})$. Then, the leaf with label u (of length ℓ) is assigned the value $v_u = x_u$. Finally, the value of the nodes are defined recursively by setting $v_u = h(v_{u0}, v_{u1})$. In words, a node's value is obtained by hashing the values of its children. The value v_λ of the root is the output of Tree.

For every string x of (unknown) polynomial length, we can bound its length by $\kappa^{\log \kappa}$, for all large enough κ. Let ℓ be the smallest integer such that $\kappa 2^\ell > \kappa^{\log \kappa}$ and let $x' = 0^{\kappa 2^\ell - |x|} x$; that is, x' is the string of length $\kappa 2^\ell$ that is obtained by padding x to the left with enough zeroes. A naive computation of $\mathsf{Tree}(x')$ would take time $\kappa 2^\ell > \kappa^{\log \kappa}$ which is too long. We show now that $\mathsf{Tree}(x')$ can actually be computed in time that is proportional to $|x|$ and not $|x'|$. The main property that enables us to do this is due to the fact that $h(0^\kappa, 0^\kappa) = 0^\kappa$. Thus, all the values of all of the nodes that are ancestors of padded blocks are just 0^κ. Let τ be the smallest integer such that $\kappa 2^\tau \geq |x|$, and pad x to the left with zeroes so that it is of size $\kappa 2^\tau$ (which is less than twice the size of x). Then, compute $\mathsf{Tree}(x)$, where x here refers to the padded version. This computation takes time that is linear in $2^\tau < |x|$. Finally, in order to compute $\mathsf{Tree}(x')$ it suffices to compute the values of the ancestors of the root of the subtree with x' in the leaves. The value of the parent of $\mathsf{Tree}(x)$ is just $h(0^k, \mathsf{Tree}(x))$ since the entire subtree to the left is zeroes. Thus, we just need to compute $v_1 = h(0^k, \mathsf{Tree}(x))$, $v_2 = (h(0^k, v_1)$, $v_3 = (h(0^k, v_2)$ and so on, up unto the root of $\mathsf{Tree}(x')$; this requires just $\ell - \tau$ hash computations. We conclude that $\mathsf{Tree}(x')$ can be computed in time that is polynomial in the length of x.

The protocol. Intuitively, the protocol proceeds as follow: P_1 and P_2 perform a binary search over their inputs to identify the first block of size κ where their inputs differ. Once they find this block, they can just exchange the values in these blocks, since they are of fixed size κ. In order to carry out this binary search, P_1 and P_2 start by conceptually padding their inputs to the left with zeroes up to size $\kappa 2^\ell$. Then, they compute Tree on the left half of their padded inputs, and compare the result. If the values are different, then this implies that there input strings differ in the left half and they throw out the right half; otherwise, they differ in the right half and so throw out the left half. They then proceed recursively until they reach the level of the leaves (this requires ℓ iterations where ℓ is upper bounded by $\log^2 \kappa$). At this point, they have found the most significant block where their inputs differ, and can exchange the block to see whose input is larger.

This protocol is size independent, as in Definition 4.5: correctness follows from the collision resistance of the hash function, computation efficiency follows from the fact that Tree can be computed on the padded input in time that is polynomial in the original input, and communication efficiency is immediate by taking the upper bound on the length of each message to be κ (this also bounds the number of rounds which is $\log^2 \kappa$).

We conclude:

Theorem 4.11 *There exists a size-independent protocol for the greater-than function. Thus, assuming the existence of collision-resistant hash functions and fully homomorphic encryption, there exists protocols for securely computing the greater-than function in classes 2.a, 2.b and 2.c, in the presence of static semi-honest and malicious adversaries.*

5 Negative Results And Separations Between Classes

In this section, we deepen our understanding of the feasibility of achieving input-size hiding by proving impossibility results for all classes where general secure computation cannot be achieved (i.e., for classes 1.b, 1.d, 2.a, 2.b and 2.c). In addition, we show that the set of functions computable in class 2.b is a strict subset of the set of functions computable in 2.a and 2.b, and that classes 1.b and 1.d are incomparable (they are not equal and neither is a subset of the other). Finally, we consider the relations between subclasses of class 1 and class 2, and show that class 2.b is a strict subset of class 1.b, but class 2.c is *not* (and so sometimes hiding both parties' inputs is easier than hiding only one party's input).

5.1 Not All Functions can be Securely Computed in Class 2

In this section we show that there exist functions for which it is impossible to achieve input-size hiding in any subclass of class 2 (where neither parties' input sizes are revealed). In order to strengthen the result, we demonstrate this on a function which has *fixed output size*. Thus, the limitation is not due to issues related to revealing the output size (as in class 2.b), but is inherent to the problem of hiding the size of the input from both parties.

The following theorem is based on the communication complexity of a function. Typically, communication complexity is defined for functions of equal sized input. We therefore generalize this definition, and measure the communication complexity of a function, as a function of the smaller of the two inputs. That is, a function f has randomized communication complexity $\Omega(g(n))$ if any probabilistic protocol for computing $f(x, y)$ with negligible error requires the parties to exchange $\Omega(g(n))$ bits, where $n = \mathsf{min}\{|x|, |y|\}$.[4]

Theorem 5.1 *Let $\mathcal{R}$ be a range of constant size, and let $f : \{0,1\}^* \times \{0,1\}^* \to \mathcal{R}$ be a function. If there exists a constant $\varepsilon > 0$ such that the randomized communication complexity of f is $\Omega(n^\varepsilon)$, then f cannot be securely computed in class 2.a, 2.b or 2.c, in the presence of static semi-honest adversaries.*

Proof: The idea behind the proof of the theorem is as follows. On the one hand, if a function has $\Omega(n^\varepsilon)$ communication complexity, then the length of the transcript cannot be independent of the

[4]Even more formally, we say that a probabilistic protocol π $\mathsf{computes}$ f if there exists a negligible function μ such that for every $x, y \in \{0,1\}^*$ the probability that the output of $\pi(x, y)$ does not equal $f(x, y)$ is at most $\mu(n)$, where $n = \mathsf{min}\{|x|, |y|\}$. Next, we say that f has $\mathsf{communication\ complexity}$ $\Omega(g(n))$ if for every protocol for computing f (as defined above) there exists a constant c and an integer $N \in \mathbb{N}$ such that for every $n > N$, the number of bits sent by the parties is at least $c \cdot g(n)$.

input lengths, and must grow as the inputs grow. On the other hand, in class 2 the input lengths are never revealed and since the output range is constant, the output says almost nothing about the input lengths. Thus, we can show that the length of the transcript must actually be independent of the input lengths, in contradiction to the assumed communication complexity of the function. We now prove this formally.

Let f be a family of functions as in the theorem statement, and assume by contradiction that there exists a protocol π that securely computes f in class 2.a. (We show impossibility for class 2.a since any protocol for class 2.b or 2.c can be converted into a protocol for class 2.a by simply having P_1 send P_2 the output at the end. Thus, impossibility for class 2.a implies impossibility for classes 2.b and 2.c as well.)

We claim that there exists a polynomial $p(\cdot)$ such that the communication complexity of π is at most $p(\kappa)$. Intuitively, this is due to the fact that the transcript cannot reveal anything about the input size and so must be bound by a fixed polynomial. Proving this formally is a little bit more tricky, and we proceed to do this now. Let $\alpha \in \mathcal{R}$ be an output value, and let $I_\alpha \subseteq \{0,1\}^* \times \{0,1\}^*$ be the set of all string pairs such that for every $(x,y) \in I_\alpha$ it holds that $f(x,y) = \alpha$. Now, by the definition of class 2.a, there exist simulators $\mathcal{S}_1$ and $\mathcal{S}_2$ that generate P_1 and P_2's views from $(x, f(x,y))$ and $(y, f(x,y))$, respectively. Thus, for every $(x,y) \in I_\alpha$, the simulators $\mathcal{S}_1$ and $\mathcal{S}_2$ must simulate given only (x,α) and (y,α), respectively.

Let x be the smallest string for which there exists a y so that $(x,y) \in I_\alpha$, and let $p'(\cdot)$ be the polynomial that bounds the running-time of $\mathcal{S}_1$. Define $p_\alpha(\kappa) = p'(|x| + |\alpha| + \kappa)$; note that this is a polynomial in κ since $|x|$ and $|\alpha|$ are constants. We claim that the polynomial $p_\alpha(\cdot)$ is an upper bound on the length of the transcript for *every* $(x,y) \in I_\alpha$. This follows immediately from the fact that $\mathcal{S}_1$ runs in time that is polynomial in its input plus the security parameter. Thus, it cannot write a transcript longer than this when given input (x,α). If the transcript upon input $(x,y) \in I_\alpha$ is longer than $p_\alpha(\kappa)$ with non-negligible probability, then this yields a trivial distinguisher, in contradiction to the assumed security with simulator $\mathcal{S}_1$.

Repeating the above for every $\alpha \in \mathcal{R}$, we have that there exists a set $P = \{p_\alpha(\kappa)\}_{\alpha\in\mathcal{R}}$ of polynomials so that any function upper bounding these polynomials is an upper bound on the transcript length for *all* inputs $(x,y) \in \{0,1\}^*$. Since $\mathcal{R}$ is of constant size, we have that there exists a single polynomial $p(\kappa)$ that upper bounds all the polynomials in P, for every κ.[5] We conclude that there exists a polynomial $p(\kappa)$ that upper bounds the size of the transcript, for all $(x,y) \in \{0,1\}^*$.

Now, let c be a constant such that $p(\kappa) < \kappa^c$, for all large enough κ. We construct a protocol π' for f as follows. On input $(x,y) \in \{0,1\}^* \times \{0,1\}^*$, execute π with security parameter $\kappa = n^{\varepsilon/2c}$, where $n = \mathsf{min}\{|x|,|y|\}$. By the correctness of π, we have that the output of $\pi(x,y)$ equals $f(x,y)$ except with negligible probability. This implies that the output of $\pi'(x,y)$ also equals $f(x,y)$ except with negligible probability (the only difference is that we need to consider larger inputs (x,y), but in any case correctness only needs to hold for all large enough inputs). Thus, π' computes f; see Footnote 4. The proof is finished by observing that the communication complexity of protocol π' is upper bounded by $p(\kappa) < (n^{\varepsilon/2c})^c = n^{\varepsilon/2}$, in contradiction to the assumed lower bound of $\Omega(n^\varepsilon)$ on the communication complexity of f. ■

Impossibility. From results on communication complexity [KN97], we have that:

[5]This argument is not true if $\mathcal{R}$ is not of a constant size. This is because it is then possible that the set of polynomials bounding the transcript sizes is $P = \{n^i\}_{i\in\mathbb{N}}$. Clearly each member of P is a polynomial; yet there is no polynomial that upper bounds all of P.

- The inner product function $\mathsf{IP}(x,y) = \sum_{i=1}^{\mathsf{min}(|x|,|y|)} x_i \cdot y_i \mod 2$ has communication complexity $\Omega(n)$.
- The set disjointness function defined by $\mathsf{DISJ}(X,Y) = 1$ if $X \cap Y = \emptyset$, and equals 0 otherwise has communication complexity $\Omega(n)$.[6] This implies that $\mathsf{INTERSECT}(X,Y) = X \cap Y$ also has communication complexity $\Omega(n)$.
- The Hamming distance function $\mathsf{HAM}(x,y) = \sum_{i=1}^{\mathsf{min}(|x|,|y|)} (x_i - y_i)^2$ has communication complexity $\Omega(n)$.

Thus:

Corollary 5.2 *The inner product, set disjointness, set intersection and Hamming distance functions cannot be securely computed in classes 2.a, 2.b or 2.c, in the presence of static semi-honest adversaries.*

Thus our protocol for set intersection in Section 4.2.2 that hides only one party's input size is "optimal" in that it is impossible to hide both parties' input sizes.

We conclude by observing that by combining Corollary 4.8 and Theorem 5.1, we obtain an almost complete characterization of the functions with constant output size that can be securely computed in class 2. This is because any function with fixed output length that can be efficiently computed with polylogarithmic communication complexity has a size-independent protocol by Definition 4.5, and so can be securely computed in all of class 2. We therefore conclude:

Corollary 5.3 *Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be a function. If f can be efficiently computed with polylogarithmic communication complexity, then it can be securely computed in all of class 2 in the presence of static semi-honest and malicious adversaries, assuming the existence of collision-resistant hash functions and fully homomorphic encryption schemes. In contrast, if there exists an $\varepsilon > 0$ such that the communication complexity of f is $\Omega(n^\varepsilon)$ then f cannot be securely computed in any subclass of class 2.*

The above corollary is not completely tight since f may have communication complexity that is neither polylogarithmic, nor $\Omega(n^\varepsilon)$. In addition, our lower and upper bounds do not hold for functions that can be inefficiently computed with polylogarithmic communication complexity.

5.2 A Separation Between Classes 2.a/c and 2.b

It is clear that any function that can be securely computed in class 2.b (where party P_1 receives $f(x,y)$, and party P_2 receives nothing at all) can be securely computed in class 2.a (where both parties receive $f(x,y)$). This is because class 2.a can be achieved by simply having the parties run a protocol for class 2.b, and then have P_1 send the output to P_2 at the end. Likewise, any function that can be securely computed in class 2.b can be securely computed in class 2.c.

In this section, we study the converse and show that there exist functions that can be securely computed in classes 2.a and 2.c but cannot be securely computed in class 2.b. Thus, we have the

[6] The disjointness function is not symmetric. However, it can be made symmetric using the method described in Footnote 3.

set of functions that can be securely computed in classes 2.a and 2.c is *strictly larger* than the set of functions that can be securely computed in class 2.b.

Let $\mathsf{vecxor} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be the function that takes two vectors of bits $\vec{x} = \langle x_1, \ldots, x_n \rangle$ and $\vec{y} = \langle y_1, \ldots, y_m \rangle$ and outputs the vector of length $\mathsf{min}(n, m)$ with the ith element being $x_i \oplus y_i$. That is, $\mathsf{vecxor}(\vec{x}, \vec{y}) = \langle x_1 \oplus y_1, \ldots, x_t \oplus y_t \rangle$, where $t = \mathsf{min}(n, m)$. We stress that there is no a priori bound on the length of the vectors

In the theorem below, we will show that vecxor can be securely computed in classes 2.a/c but not in class 2.b. The fact that it can be computed in classes 2.a/c follows from the fact that the min function can be securely computed in classes 2.a/c, as we have already shown in Corollary 4.10, and once t is known it is not hard to securely compute $\langle x_1 \oplus y_1, \ldots, x_t \oplus y_t \rangle$. In order to prove that vecxor *cannot* be securely computed in class 2.c, we first show that the length of the transcript in vecxor must be bounded by a fixed polynomial. Intuitively, this is due to the fact that only party P_1 receives output and so cannot tell whether P_2 has a vector of the same length or longer. In addition, P_2 receives no information and so cannot know anything about the length of P_1's vector. Combining these together we obtain that the transcript length must be independent of the parties' input lengths, and in particular bound by a fixed polynomial. Next, we show that vecxor can actually be used to transmit an arbitrary string of arbitrary length from P_2 to P_1. This works by setting P_1's input to be the string of all zeroes and P_2's input to be the string that it wishes to transmit. By the definition of vecxor, the output is just P_2's input string. Combining the above, we have that vecxor can be used to transmit an arbitrarily long string using a fixed polynomial number of bits, which is impossible.

Theorem 5.4 *The function* vecxor *defined above cannot be securely computed in class 2.b in the presence of static semi-honest adversaries, but assuming the existence of fully homomorphic encryption with circuit privacy it can be securely computed in classes 2.a and 2.c in the presence of static semi-honest adversaries.*

Proof: In Section 4.3.3 we showed that the function min can be securely computed in classes 2.a and 2.c. We now show that vecxor can also be securely computed in class 2.a, in the presence of static semi-honest adversaries. First, P_1 and P_2 use a class 2.a protocol to compute $t = \mathsf{min}(n, m)$; recall that both parties learn t. Then, given t, parties P_1 and P_2 can send each other the vector $\langle x_1, \ldots, x_t \rangle$ and $\langle y_1, \ldots, y_t \rangle$, respectively, and then each can locally compute $\langle x_1 \oplus y_1, \ldots, x_t \oplus y_t \rangle$ and output it. It is clear that both parties learn nothing more than the output. In order to compute vecxor in class 2.c, the protocol is the same except that only P_2 sends P_1 the vector $\langle y_1, \ldots, y_t \rangle$, and only P_1 obtains output. Observe that in class 2.c, party P_2 receives $1^{|f(x,y)|}$ which is exactly 1^t, and so having P_2 receive t from the computation of $\mathsf{min}(n, m)$ is not a problem.

We now show that it is impossible to securely compute vecxor in class 2.b. Assume, by contradiction, that there exists a protocol π that securely computes vecxor in Class 2.b. Let $T(\kappa, \vec{x}, \vec{y})$ be the random variable representing the number of bits exchanged by honest P_1 and P_2 when running π with respective inputs $\vec{x}$ and $\vec{y}$, and security parameter κ (where the probability is taken over the random tapes of P_1 and P_2).

Let $\vec{v}_\emptyset$ denote the empty vector of length 0, and let c be a constant such that $T(\kappa, \vec{v}_\emptyset, \vec{v}_\emptyset) \leq \kappa^c$ for all large enough κ (such a constant exists since the protocol runs in polynomial time). Now, for every vector $\vec{y}$ it holds that $\mathsf{vecmin}(\vec{v}_\emptyset, \vec{v}_\emptyset) = \mathsf{vecmin}(\vec{v}_\emptyset, \vec{y}) = \vec{v}_\emptyset$. This implies that for every vector $\vec{y}$ it must hold that $T(\kappa, \vec{v}_\emptyset, \vec{y}) \leq \kappa^c$ for all large enough κ. Otherwise, a distinguisher in the case of a corrupted P_1 with input $\vec{v}_\emptyset$ can distinguish the case that P_2 has $\vec{v}_\emptyset$ or $\vec{y}$ just by looking at the number of bits exchanged. (Recall that P_1 receives the output only, which in this case is just $\vec{v}_\emptyset$.)

Now let $\vec{y}^* = \langle y_1, \ldots, y_t \rangle$ be a vector of t random bits, where $t = \omega(\kappa^c)$. Furthermore, let $\vec{x}^* = \langle 0, \ldots, 0 \rangle$ of length t as well (i.e., $\vec{x}$ is the all-zero vector of length-t). As we have just argued, $T(\kappa, \vec{v}_\emptyset, \vec{y}^*) \leq \kappa^c$ for all large enough κ. In addition, since P_2 receives no output in any case, it must also hold that $T(\kappa, \vec{x}^*, \vec{y}^*) \leq \kappa^c$ for infinitely many κ; otherwise, a distinguisher in the case that P_2 is corrupted can distinguish between the case that P_1 had input $\vec{v}_\emptyset$ and the case that it had input $\vec{x}^*$.

The proof is concluded by observing that the output of P_1 from an execution of $\mathsf{vecxor}(\vec{x}^*, \vec{y}^*)$ is $\vec{y}^*$, and therefore the protocol π can be used to have P_2 send a random t-bit string to P_1. However, $t = \omega(\kappa^c)$ whereas the number of bits exchanged in $\mathsf{vecxor}(\vec{x}^*, \vec{y}^*)$ is less than κ^c, in contradiction to the fact that it is impossible to compress a random string of length $\omega(\kappa^c)$ to a string of length κ^c. ■

We conclude that:

Corollary 5.5 *Assuming the existence of fully homomorphic encryption with circuit privacy, the set of functions that can be securely computed in class 2.b is a* strict subset *of the set of functions that can be securely computed in both class 2.a and class 2.c.*

5.3 Not All Functions can be Securely Computed in Class 1.b and 1.d

In classes 1.a/c/e, even if a party does not learn the output it still learns the *size* of the output. This enables us to use fully homomorphic encryption in order to securely compute any function in these classes. The fact that both parties learn the output size seems inherent in this method since one party evaluates the circuit inside the encryption (and so sees an encryption of the output, revealing its size) and the other receives the actual output. In this section, we show that it is indeed inherent. Specifically, we show that it is impossible to obtain an analogous feasibility result for classes 1.b and 1.d; recall that in these classes the party that does *not* learn the output also does *not* learn the output size.

The impossibility result for class 1.b resembles the result for class 2.b in the previous section and is based on the impossibility of compressing random strings. In contrast, the impossibility for class 1.d is very different. In order to see why, observe that in class 1.d party P_2 learns both $f(x, y)$ and $1^{|x|}$. Thus, without taking security into account, party P_1 can just send its input to party P_2 who can compute $f(x, y)$ and be finished. We therefore cannot use information-theoretic communication complexity and incompressibility arguments. We will still use an incompressibility based argument, but one that is based on computational entropy.

Impossibility for class 1.b (where $\boldsymbol{P_1}$ receives $\boldsymbol{f(x, y)}$ and $\boldsymbol{P_2}$ receives $\boldsymbol{1^{|x|}}$). We prove impossibility here for a size-hiding version of oblivious transfer, where P_1 is the receiver in the oblivious transfer and has input a bit $x \in \{0, 1\}$, and P_2 has two strings (y_0, y_1) of arbitrary length. The output is the string y_x.[7] Recall that in class 1.b, party P_1 receives the output and party P_2 only receives the length of P_1's input, which in this case is known (because it is a single bit). Formally, define $\mathsf{OT}(x, (y_0, y_1)) = y_x$.

[7]The OT function is not symmetric, but can be made symmetric using the method described in Footnote 3. Note that for proving impossibility, it suffices to consider the asymmetric case, or in the notation of Footnote 3 it suffices to prove impossibility when $b_1 = 0$ and $b_2 = 1$.

Intuitively, OT can be used to have P_2 communicate many bits to P_1. Furthermore, since P_2 cannot learn the output length, the length of the transcript must be independent of the output length, which is the number of bits transmitted. Thus, as in the proof of Theorem 5.4, this contradicts the fact that it is impossible to transmit arbitrary long strings using fewer bits.

Theorem 5.6 *The unbounded oblivious transfer function* OT *cannot be securely computed in class 1.b, in the presence of static semi-honest adversaries.*

Proof: Assume by contradiction that there exists a protocol π that securely computes OT in class 1.b. Let $T(\kappa, x, Y)$ be the random variable representing the number of bits exchanged by honest P_1 and P_2 when running π with respective inputs x and (y_1, y_2) and security parameter κ (where the probabilities is taken over the random tapes of P_1 and P_2). Let $x^* = 0$ and $y_0^* = y_1^* = 0$ (i.e., all inputs are single bits). Let c be the integer such that $T(\kappa, x^*, (y_0^*, y_1^*)) \leq \kappa^c$, for all large enough κ (this follows from the fact that the protocol is polynomial time).

Next, let y_1' be a random string of length $m = \omega(\kappa^c)$. Note that $\mathsf{OT}(x^*, (y_0^*, y_1')) = \mathsf{OT}(x^*, (y_0^*, y_1^*)) = 0$, since $x^* = 0$ and $y_0^* = 0$. Therefore, it must also hold that $T(\kappa, x^*, (y_0^*, y_1')) \leq \kappa^c$ for all large enough κ, or a corrupted P_1 could distinguish the case that P_2 has input (y_0^*, y_1') or input (y_0^*, y_1^*) by the length of the transcript. Thus, increasing the size of P_2's input cannot increase the size of the transcript. Now, let $x' = 1$. By the function definition, $\mathsf{OT}(x', (y_0^*, y_1')) = y_1'$. However, as before, it must hold that $T(\kappa, x', (y_0^*, y_1')) \leq \kappa^c$ for all large enough κ, or a corrupted P_2 could distinguish the case that P_1 has x_1 or x_2 (note that P_2 receives the length of P_1's input but this is just a single bit in both cases).

By the fact that $\mathsf{OT}(x', (y_0^*, y_1')) = y_1'$ and $T(\kappa, x', (y_0^*, y_1')) = \mathcal{O}(\kappa^c)$, we have that the protocol π can be used to have P_2 send a random string y_1' of length $m = \omega(\kappa^c)$ to P_1, while communicating only $\mathcal{O}(\kappa^c)$ bits, in contradiction to the incompressibility of random strings. ■

Impossibility for class 1.d (where P_1 receives nothing and P_2 receives $(1^{|x|}, f(x, y))$). As we have mentioned above, when proving impossibility for class 1.d, we cannot use information-theoretic lower bounds in communication complexity and arguments related to incompressibility of random strings. Rather, the argument here will be more "cryptographic" in style.

Let $F = \{F^\kappa\}$ be a pseudorandom function family, where for every $\kappa \in \mathbb{N}$ we have $F^\kappa : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^\kappa$. In addition, let omprf be the oblivious multi-point PRF-evaluation function that takes a string $x \in \{0,1\}^\kappa$ from P_1 and a set of m strings $Y = \{y_i \in \{0,1\}^\kappa\}_{i=1}^m$ from P_2, and outputs the set $Z = \{z_i = F_x^\kappa(y_i)\}_{i=1}^m$. That is, the function evaluates the pseudorandom function with key x on all the points y_i. Recall that in class 1.d, party P_2 receives the output, and P_1 must receive nothing.[8]

Intuitively, omprf cannot be securely computed in class 1.d because the output of P_2 has high computational entropy (indeed, P_2 cannot distinguish the values in its output from truly random). Furthermore, P_2 can compute its output from its view alone. Now, if the transcript of the execution is *independent* of the output size (as we will show it must be since P_1 cannot learn the P_2's input size or the output size), then this means that P_2 can compute a large set of pseudorandom strings from a short view (although P_2's input is not short, it can be fixed and this makes no difference). By the pseudorandomness property, this means that P_2 can compute a large set of random strings from a short view, and this is impossible by an incompressibility argument.

[8]As with the OT function, omprf is not symmetric but this suffices; see Footnote 7.

Theorem 5.7 *Assuming the existence of one-way functions, the function* omprf *defined above cannot be securely computed in class 1.d, in the presence of static semi-honest adversaries.*

Proof: Assume by contradiction that there exists a protocol π for securely computing omprf in class 1.d, and let $T(\kappa, x, Y)$ be the random variable representing the number of bits exchanged in an execution, as in previous proofs. Let $c \in \mathbb{N}$ be a constant such that $T(\kappa, x, \emptyset) \leq \kappa^c$; such a constant exists since π is a polynomial-time protocol (and x is always of size κ here). Then, for any Y of size $\omega(\kappa^c)$ it must also hold that $T(\kappa, x, Y) < \kappa^c$; otherwise a distinguisher in the case of a corrupted P_1 can distinguish the case that P_1 has input $\emptyset$ and input Y.

By the assumption that π is secure in the presence of semi-honest adversaries, there exists a simulator $\mathcal{S}_2$ that on input Y, 1^κ and $Z = \{F_x^\kappa(y)\}_{y \in Y}$ produces the view of P_2 in a protocol interaction with P_1, upon inputs x and Y. In particular, this means that given Y, the simulated random tape and the simulated received messages, it is possible to efficiently compute the output set Z. Using a standard argument, this should also work if P_2's random tape is pseudorandom, and not truly random one (i.e., if P_2 generates its random tape using a pseudorandom generator and a seed of length κ).

We now use $\mathcal{S}_2$ to distinguish between a pseudorandom function family $F = \{F^\kappa\}$ and a truly random function family $R = \{R^\kappa\}$. The distinguisher $\mathcal{D}$ with access to a function f (that is either F^κ or R^κ) gives $\mathcal{S}_2$ the tuple $(Y, 1^\kappa, Z)$ where $Z = \{f(y)\}_{y \in Y}$ and obtains back the view of P_2 in a protocol execution. Then, $\mathcal{D}$ uses the view generated by $\mathcal{S}_2$ in order to recompute Z using P_2's instructions in the protocol (since P_2 must output Z in a protocol execution, it can compute this from its view). Finally, $\mathcal{D}$ outputs 1 if and only if $Z = \{f(y)\}_{y \in Y}$.

If $f = F^\kappa$ is a pseudorandom function, then $\mathcal{D}$ outputs 1 except with negligible probability. Otherwise the output of $\mathcal{S}_2$ can be easily distinguished from the view of P_2 in a real execution (where P_2 outputs the correct Z except with negligible probability). In contrast, if $f = R^\kappa$ is a truly random function, then due to incompressibility of random data, it is impossible to efficiently compute $Z = \{R^\kappa(y)\}_{y \in Y}$ from the view output by $\mathcal{S}_2$ since this view is of length κ^c and Z is of length $\omega(\kappa^c)$. This contradicts the pseudorandomness of F, which follows from the existence of one-way functions. ■

5.4 Separation Between Classes 1.b and 1.d

We show that classes 1.b and 1.d are *incomparable*. We begin by showing that the OT function of Section 5.3 that cannot be securely computed in class 1.b *can* be securely computed in class 1.d. This is actually a triviality since in class 1.d, we have that P_2 receives the output y_x and not P_1. Thus, if $y_0 \neq y_1$ then P_2 learns P_1's actual input bit (by just looking at which string was output) and so P_1 could actually just send x to P_2. The only problem is therefore if $y_0 = y_1$, in which case P_2 should learn nothing. We solve this problem by just having P_1 and P_2 run a regular oblivious transfer protocol with all inputs being bits. P_1 plays the sender in the oblivious transfer and inputs the pair $(0, x)$, and P_2 plays the receiver and inputs 0 if $y_0 = y_1$, and inputs 1 otherwise. (Thus, if $y_0 = y_1$ then P_2 just obtains 0 and has learned nothing, and if $y_0 \neq y_1$ then P_2 learns the actual value of x.) Finally, if $y_0 = y_1$ then party P_2 just outputs y_0 (it makes no difference), and if $y_0 \neq y_1$ then P_2 obtained x and so can output y_x. Clearly, P_1 learns nothing in this protocol, while P_2 learns the output only; a simulator for this protocol can be constructed easily. We conclude that OT can be securely computed in class 1.d (assuming regular oblivious transfer, which also follows from the existence of FHE).[9]

Next, we show that the function omprf of Section 5.3 that cannot be securely computed in class 1.d *can* be securely computed in class 1.b. This can be achieved by having P_1 sample a key-pair (pk, sk) from a fully homomorphic encryption scheme with circuit privacy, and having P_1 send $\mathsf{Enc}_{pk}(x)$ to P_2. Party P_2 then uses Eval to compute $\mathsf{Enc}_{pk}(F_x^\kappa(y_i))$ for every $y_i \in Y$, and sends the results back to P_1, who can then decrypt and obtain the output. Clearly P_2 learns nothing more than $1^{|x|}$ and P_1 learns nothing more than $f(x, y)$.[10]

We have proven that classes 1.b and 1.d are incomparable. However, observing that both the OT and omprf functions are *not* output-length bounded (in the sense of Section 4.2.2 meaning that the output is bounded by a fixed polynomial in P_1's input size), we have also proven that there exist functions with unbounded output length that can be securely computed in both of these classes. Observe that this is in contrast to the result of Section 4.2.2 which proved feasibility for these classes for all output-length bounded functions. We therefore conclude:

Corollary 5.8 *Assuming the existence of fully homomorphic encryption with circuit privacy, there exist functions that can be securely computed in class 1.b but cannot be securely computed in class 1.d, and there exist functions that can be securely computed in class 1.d but cannot be securely computed in class 1.b, in the presence of static semi-honest adversaries. In addition, there exist functions with unbounded output-length that can be securely computed in each of classes 1.b and 1.d.*

5.5 Separations between Classes 1 and 2

Class 2.b is a strict subset of classes 1.b. It is straightforward to see that the function vecxor defined in Section 5.2 can be securely computed in class 1.b. This is due to the fact that in class 1.b, party P_2 may learn the length of P_1's input. Thus, P_1 can send its input length n to P_2. Then, P_2 computes $t = \mathsf{min}(n, m)$ and sends P_1 the vector $\langle y_1, \ldots, y_t \rangle$ who outputs $\langle x_1 \oplus y_1, \ldots, x_t \oplus y_t \rangle$. However, by Theorem 5.4, vecxor cannot be securely computed in class 2.b. Combining this with the trivial fact that any function that can be securely computed in class 2.b can be securely computed in class 1.b, we have the following corollary:

Corollary 5.9 *Assuming the existence of fully homomorphic encryption with circuit privacy, the set of functions that can be securely computed in class 2.b is a* strict subset *of the set of functions that can be securely computed in class 1.b.*

Class 2.c is not a subset of classes 1.b/d. By Corollary 5.2, the inner-product function cannot be securely computed in any class 2. However, since this is a function with bounded output length

[9]Recall that we focus on symmetric functions in this paper. Thus, in order to be consistent we need to show how to securely computed the symmetric version OT as described in Footnotes 3 and 7. Specifically, we need to show that it is also possible to securely compute OT in class 1.d, where party P_1 holds a pair of unbounded strings (x_0, x_1) and party P_2 holds a choice bit $y \in \{0, 1\}$. However, this is easy since P_2 can choose an FHE key-pair and send $\mathsf{Enc}_{pk}(y)$ to P_1, who can use Eval to compute $\mathsf{Enc}_{pk}(x_y)$, padded to the length that is the maximum of $|x_0|$ and $|x_1|$. This ciphertext is then sent back to P_2 who can decrypt and obtain the output. Since P_2 anyway receives the size of P_1's input in class 1.d, we are allowed to have P_1 send P_2 the padded output.

[10]The symmetric version of omprf can also be computed in class 1.b by just having P_1 send $c_1 = \mathsf{Enc}_{pk}(x_1), \ldots, c_n = \mathsf{Enc}_{pk}(x_n)$ to P_2 where $X = \{x_1, \ldots, x_n\}$, who then uses Eval to compute $\mathsf{Enc}_{pk}(F_y^\kappa(x_i))$ for every i and then send the results back to P_1 to decrypt. Since P_2 can learn the size of P_1's input in class 1.b, this is secure.

(a single bit), by Corollary 4.4 it can be securely computed in classes 1.b and 1.d. Thus, class 2.c does not include all of classes 1.b and 1.d.

In addition, both the OT and omprf functions from Section 5.3 that cannot be securely computed in classes 1.b and 1.d, respectively, *can* be securely computed in class 2.c. This can be shown using techniques similar to Protocol 4.6. It is also possible to show that omprf can be securely computed in class 2.b (because it can be computed in class 1.b and the input of P_1 in omprf is of known size). We therefore conclude:

Corollary 5.10 *Assume that fully homomorphic encryption with circuit privacy exists. Then, there exist functions that cannot be securely computed in classes 1.b and 1.d in the presence of static semi-honest adversaries, but can be securely computed in class 2.c. In addition, there exist functions that cannot be securely computed in class 1.d but can be securely computed in class 2.b.*

This corollary is somewhat surprising since it means that it is sometimes possible to compute while hiding both parties' inputs but not while hiding one party's input (of course, depending on who receives the output and output length).

6 Summary

Our work provides quite a complete picture of feasibility, at least on the level of in which classes can all functions be securely computed and in which not. In addition, we show separations between many of the subclasses, demonstrating that the input-size hiding landscape is rich. In Table 1 we provide a summary of what functions can and cannot be computed in each class. This is in no terms a full characterization, but rather some examples that demonstrate the feasibility and infeasibility in the classes.

	All f (bounded output)	**All f** (even unbounded output)	GT ($x > y$)	vecxor	Intersection	OT	omprf
2.a	×	×	✓	✓	×	✓	✓
2.b	×	×	✓	×	×	×	✓
2.c	×	×	✓	✓	×	✓	✓
1.a	✓	✓	✓	✓	✓	✓	✓
1.b	✓	×	✓	✓	✓	×	✓
1.c	✓	✓	✓	✓	✓	✓	✓
1.d	✓	×	✓	✓	✓	✓	×
1.e	✓	✓	✓	✓	✓	✓	✓

Figure 1: Summary of feasibility

References

[ACT11] Giuseppe Ateniese, Emiliano De Cristofaro, and Gene Tsudik. (if) size matters: Size-hiding private set intersection. In *Public Key Cryptography*, pages 156–173, 2011.

[Bea91] Donald Beaver. Foundations of secure interactive computing. In *CRYPTO*, pages 377–391, 1991.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. *FOCS*, 2011.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[CV12] Melissa Chase and Ivan Visconti. Secure database commitments and universal arguments of quasi knowledge. In *CRYPTO*, pages 236–254, 2012.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[GL90] Shafi Goldwasser and Leonid A. Levin. Fair computation of general functions in presence of immoral majority. In *CRYPTO*, pages 77–93, 1990.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

[Gol04] Oded Goldreich. *Foundations of Cryptography Volume 2, Basic Applications.* Cambridge University Press, 2004.

[HL10] Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols: Techniques and Constructions.* Springer-Verlag, 2010.

[IP07] Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In *TCC*, pages 575–594, 2007.

[KN97] Eyal Kushilevitz and Naom Nisan. *Communication Complexity.* Cambridge University Press, 1997.

[MR91] Silvio Micali and Phillip Rogaway. Secure computation (abstract). In *CRYPTO*, pages 392–404, 1991.

[MRK03] Silvio Micali, Michael O. Rabin, and Joe Kilian. Zero-knowledge sets. In *FOCS*, pages 80–91. IEEE Computer Society, 2003.

[Nis93] Naom Nisan. The communication complexity of threshold gates. *Combinatorics, Paul Erdos is Eighty*, 1:301–315, 1993.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

A Tools – Threshold Fully-Homomorphic Encryption

Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be a tuple of efficient, possibly randomized algorithms, where

- $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$ is a randomized key generation algorithm, that outputs a public key and a secret key;
- $c \leftarrow \mathsf{Enc}_{pk}(m)$ is a randomized encryption algorithm (both the plaintext and the ciphertext space are defined by the public key pk). We write $c \leftarrow \mathsf{Enc}_{pk}(m; r)$ when we want to make the randomness used during the encryption explicit;
- $m' \leftarrow \mathsf{Dec}_{sk}(c)$ outputs a message from the plaintext space or a special symbol $\perp$;
- $\overline{c} \leftarrow \mathsf{Eval}_{pk}(\mathcal{C}, \langle c_1, \ldots, c_n \rangle)$ takes as input a circuit $\mathcal{C}$ defined for n inputs and a tuple of n ciphertexts $\langle c_1, \ldots, c_n \rangle$, and outputs a new ciphertext $\overline{c}$.

We need circuit privacy in our application; since this implies compactness we do not include compactness in the definition.

Definition A.1 (Fully-Homomorphic Scheme) *We say that* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *is a* fully-homomorphic encryption scheme with circuit privacy *if* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is a public-key encryption scheme that is secure in the presence of chosen-plaintext attacks, and the following holds:*

- Correctness: *For all polynomial-size circuits $\mathcal{C}$ and messages $m_1, \ldots, m_n$ with $n = \mathrm{poly}(\kappa)$:*

$$\Pr\Big[\mathsf{Dec}_{sk}(\mathsf{Eval}_{pk}(\mathcal{C}, \langle \mathsf{Enc}_{pk}(m_1), \ldots, \mathsf{Enc}_{pk}(m_n)\rangle)) \neq \mathcal{C}(m_1, \ldots, m_n)\Big] \leq \mathsf{negl}(\kappa)$$

  where the probability is taken over the random coins of all the algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$.

- Circuit Privacy: *For every circuit $\mathcal{C}$, every (pk, sk) in the range of* $\mathsf{Gen}(1^\kappa)$ *and every tuple of n ciphertexts $\langle c_1 = \mathsf{Enc}_{pk}(m_1), \ldots, c_n = \mathsf{Enc}_{pk}(m_n)\rangle$, the following distributions (over the random coins of* Eval *and of* Enc *encrypting $\mathcal{C}(m_1, \ldots, m_n)$) are statistically close*

$$\big\{\mathsf{Eval}_{pk}(\mathcal{C}, \langle c_1, \ldots, c_n \rangle)\big\} \quad \text{and} \quad \big\{\mathsf{Enc}_{pk}(\mathcal{C}(m_1, \ldots, m_n))\big\}.$$

In some of the constructions we will need to use a threshold version of FHE. We will use two functionalities for shared key generation and shared decryption:

- Let $((sk_1, pk), (sk_2, pk)) \leftarrow \mathsf{ThrGen}(1^\kappa, 1^\kappa)$ be the randomized functionality that takes input 1^κ from P_1 and P_2 and computes $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$. The functionality then chooses a random $r \in \{0,1\}^\kappa$ and defines the shares of sk by $sk_1 = r$ and $sk_2 = sk \oplus r$. Finally, it gives the outputs (sk_1, pk) to P_1 and (sk_2, pk) to P_2.
- Let $(m, \lambda) \leftarrow \mathsf{ThrDec}((C_A, sk_1), (C_B, sk_2))$ be the functionality that computes and outputs $m = \mathsf{Dec}_{sk_1 \oplus sk_2}(C_A)$ if $C_A = C_B$, and $\perp$ otherwise.

We remark that when using threshold decryption, computational circuit privacy is implied from the regular CPA security of the fully homomorphic encryption scheme. Thus, circuit privacy (which as we have defined it is *statistical*), does not to be required as well.

State of the art: The first construction of fully homomorphic encryption was given by Gentry in [Gen09]. Right after this seminal work, many optimization of Gentry's protocol have been proposed. Perhaps most interestingly, Brakerski and Vaikuntanathan [BV11] showed that fully homomorphic encryption can be instantiated based solely on standard assumptions (i.e., LWE). When considering the threshold version of FHE, we are not aware of any FHE scheme that "naturally" supports threshold key generation and threshold decryption. However, since FHE implies semi-honest oblivious transfer and one-way functions, it is possible to use the protocols of [GMW87, Yao86] to securely implement Class 0 protocols for $\mathsf{ThrGen}, \mathsf{ThrDec}$.

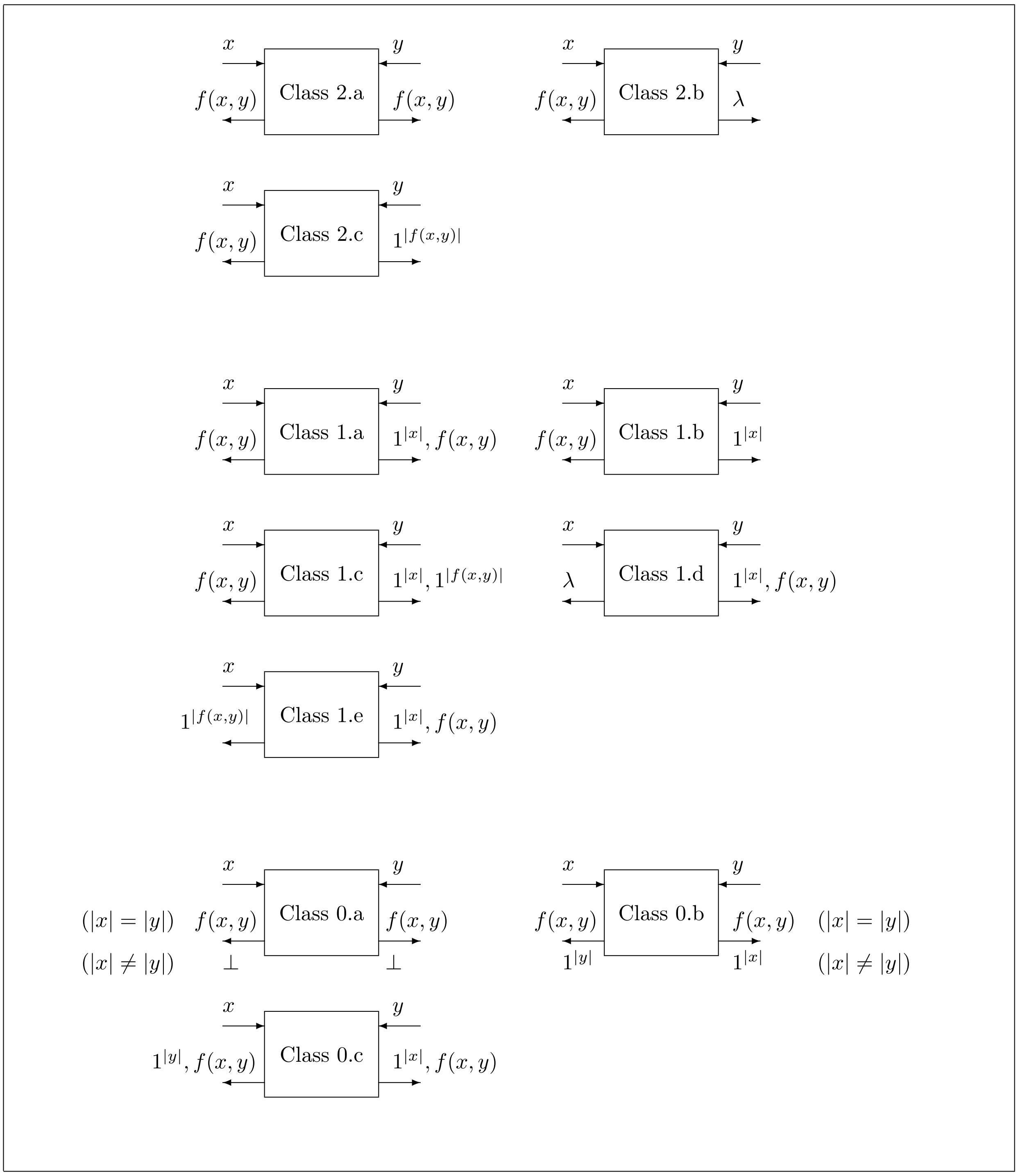

Figure 2: Classification of Input-Size Hiding Ideal Models