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

Yehuda Lindell**, Kobbi Nissim***, and 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.

* This work was funded by the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 239868.

** 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.

– 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

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 the full version 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 com-

plexity 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.

2 Technical Overview

In this section we provide a brief overview of the results and the techniques used through the paper. Due to space limitation, much of the technical material has been removed from this version, but can be found in the full version of this article [LNO12].

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 6 on the last page of this submission. 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. 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.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. Due to space limitations all formal definitions are deferred to the full version.

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: 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 the full version 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 3 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

As opposed to class 1, we do not have any general positive result for class 2. Indeed, in Theorem 6 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 the full version 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. Furthermore 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.

In the full version, we review the standard way that input sizes are dealt with and observe that there are important subtleties here which are typically ignored. We present the different classes of size-hiding here. The formal definitions of security based on these classes, including the ideal and real model descriptions, and the definitions for security in the presence of semi-honest and malicious adversaries, are deferred to the full version. We stress that a number of technical subtleties do arise when formalizing these notions.

3.1 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 the full version);
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 (that can be found in the full version). 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 the full version,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 6 (at the last page of this submission) 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[1].

[1] The non-symmetric case is not so different with respect to feasibility: e.g., 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). Nevertheless, it can be made symmetric by defining $f((x,b_1),(y,b_2))$ 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. 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.

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 the full version 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 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 $y = y_1, \ldots, y_m$.

Theorem 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*

PROTOCOL 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.

circuit privacy, then Protocol 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. ■

Extensions. 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 analog to the result of [ACT11] without relying on random oracles. These extensions appear in the full version.

4.2 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.

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 3 (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 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.

Due to lack of space in this abstract, we describe protocol here informally and refer to the full version for a formal description and proof. 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 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). In the full version, we formally prove the following corollary:

Corollary 4 *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*

[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.

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.

In addition, we show the following applications of the above corollary:

Corollary 5 *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 addition, the* min *function can be securely computed in classes 2.a and 2.c, in the presence of static semi-honest 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).

Due to lack of space, we present only the proof of impossibility for class 2; this provides the flavor of all of our impossibility results. All the other results can be found in the the full version.

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 $n = \mathsf{min}\{|x|,|y|\}$.[3]

[3] Even more formally, we say that a probabilistic protocol π computes f if there exists a negligible function μ such that for every $x, y \in \{0,1\}^*$ the probability that the

Theorem 6 *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 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 $(x,y) \in \{0,1\}^*$.

---

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 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 $c \cdot g(n)$.

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.[4] 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 3. 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:

- 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 $\Omega(n)$.[5] 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 7 *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 (see the full version) 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 and Theorem 6, 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 3, and so can be securely computed in all of class 2. We therefore conclude:

[4] 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.

[5] The disjointness function is not symmetric. However, it can be made symmetric using the method described in Footnote 1.

Corollary 8 *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.

Additional results. In the full version, we prove a series of impossibility results and study the relations between the different classes. Amongst other things, we show that the *oblivious transfer function* with strings of unbounded length *cannot* be securely computed in classes 1.b and 2.b, but *can* be securely computed in classes 2.a,2.c and 1.d (it can be computed in classes 1.a/c/e since all functions can be securely computed in these classes).

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	✓	✓	✓	✓	✓	✓	✓

Fig. 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.

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

[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.

[LNO12] Yehuda Lindell, Kobbi Nissim, and Claudio Orlandi. Hiding the input-size in secure two-party computation. Cryptology ePrint Archive, Report 2012/679, 2012. `http://eprint.iacr.org/`.

[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.

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

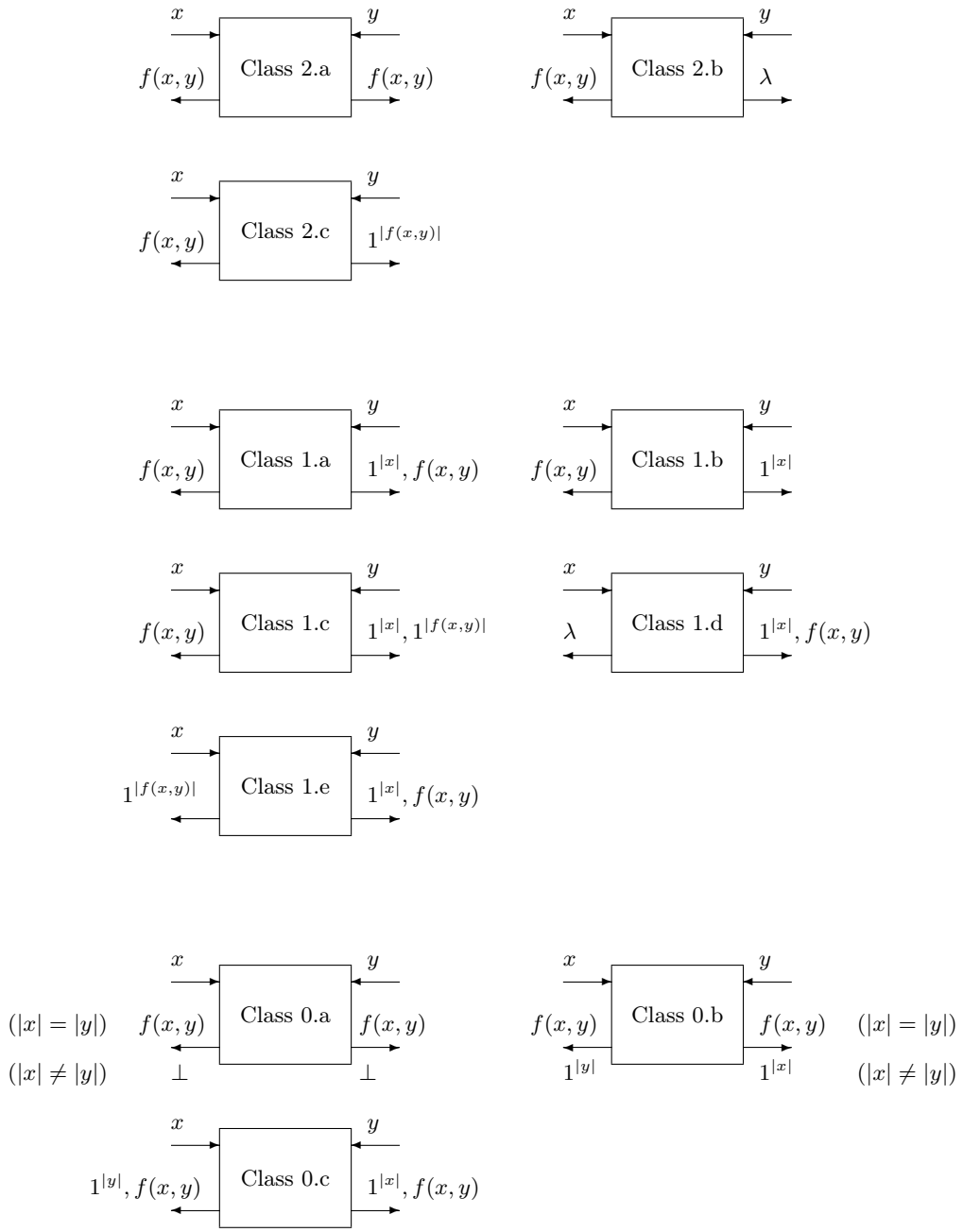

Fig. 2. Classification of Input-Size Hiding Ideal Models